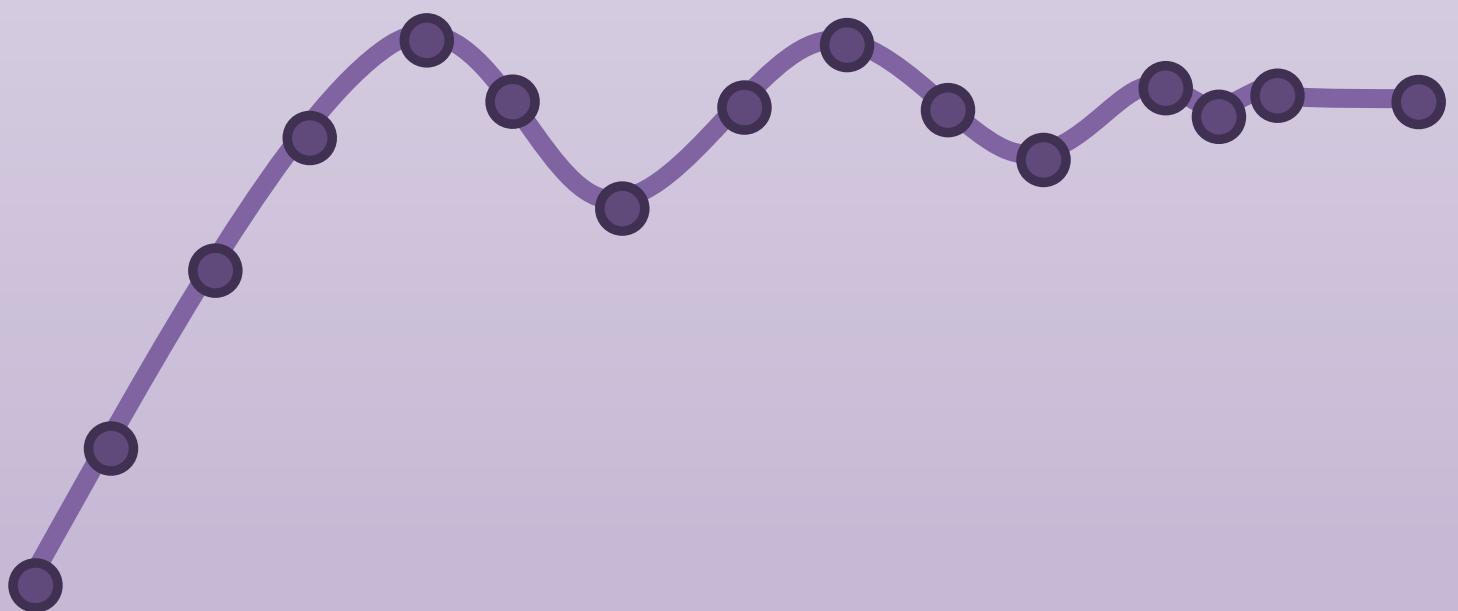


# Python for Control Engineering

Hans-Petter Halvorsen



<https://www.halvorsen.blog>



Python for Control Engineering

# Python for Control Engineering

Hans-Petter Halvorsen

2020

# Python for Control Engineering

©Hans-Petter Halvorsen

September 9, 2020



# 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 used 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 L<sup>A</sup>T<sub>E</sub>X using Overleaf.

L<sup>A</sup>T<sub>E</sub>Xi<sup>s</sup> a document preparation system used for the communication and publication of scientific documents.

For more information about L<sup>A</sup>T<sub>E</sub>X:  
<https://www.latex-project.org>

Overleaf is a web-bases L<sup>A</sup>T<sub>E</sub>Xsystem, meaning you can write your L<sup>A</sup>T<sub>E</sub>Xdocuments 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.

## **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/softwareengineering/>

<h2>Python Programming</h2> <p>Hans-Petter Halvorsen</p>  <p><a href="https://www.halvorsen.blog">https://www.halvorsen.blog</a></p>	<h2>Python for Science and Engineering</h2> <p>Hans-Petter Halvorsen</p>  <p><a href="https://www.halvorsen.blog">https://www.halvorsen.blog</a></p>
<h2>Python for Control Engineering</h2> <p>Hans-Petter Halvorsen</p>  <p><a href="https://www.halvorsen.blog">https://www.halvorsen.blog</a></p>	<h2>Python for Software Development</h2> <p>Hans-Petter Halvorsen</p>  <p><a href="https://www.halvorsen.blog">https://www.halvorsen.blog</a></p>

# Contents

<b>1 Getting Started with Python</b>	<b>13</b>
<b>1 Introduction</b>	<b>14</b>
1.1 The New Age of Programming . . . . .	14
1.2 MATLAB . . . . .	18
<b>2 What is Python?</b>	<b>20</b>
2.1 Introduction to Python . . . . .	20
2.1.1 Interpreted vs. Compiled . . . . .	21
2.2 Python Packages . . . . .	22
2.2.1 Python Packages for Science and Numerical Computations	23
2.3 Anaconda . . . . .	23
2.4 Python Editors . . . . .	24
2.4.1 Python IDLE . . . . .	24
2.4.2 Visual Studio Code . . . . .	25
2.4.3 Spyder . . . . .	25
2.4.4 Visual Studio . . . . .	25
2.4.5 PyCharm . . . . .	25
2.4.6 Wing Python IDE . . . . .	26
2.4.7 Jupyter Notebook . . . . .	26
2.5 Resources . . . . .	26
2.6 Installing Python . . . . .	26
2.6.1 Python Windows 10 Store App . . . . .	27
2.6.2 Installing Anaconda . . . . .	27
2.6.3 Installing Visual Studio Code . . . . .	27
<b>3 Start using Python</b>	<b>29</b>
3.1 Python IDE . . . . .	29
3.2 My first Python program . . . . .	29
3.3 Python Shell . . . . .	30
3.4 Running Python from the Console . . . . .	30
3.4.1 Opening the Console on macOS . . . . .	31
3.4.2 Opening the Console on Windows . . . . .	32
3.4.3 Add Python to Path . . . . .	32
3.5 Scripting Mode . . . . .	34
3.5.1 Run Python Scripts from the Python IDLE . . . . .	34
3.5.2 Run Python Scripts from the Console (Terminal) macOS . . . . .	35
3.5.3 Run Python Scripts from the Command Prompt in Windows . . . . .	36

3.5.4	Run Python Scripts from Spyder . . . . .	36
<b>4</b>	<b>Basic Python Programming</b>	<b>39</b>
4.1	Basic Python Program . . . . .	39
4.1.1	Get Help . . . . .	39
4.2	Variables . . . . .	39
4.2.1	Numbers . . . . .	41
4.2.2	Strings . . . . .	42
4.2.3	String Input . . . . .	43
4.3	Built-in Functions . . . . .	43
4.4	Python Standard Library . . . . .	44
4.5	Using Python Libraries, Packages and Modules . . . . .	45
4.5.1	Python Packages . . . . .	47
4.6	Plotting in Python . . . . .	47
4.6.1	Subplots . . . . .	50
4.6.2	Exercises . . . . .	52
<b>II</b>	<b>Python Programming</b>	<b>53</b>
<b>5</b>	<b>Python Programming</b>	<b>54</b>
5.1	If ... Else . . . . .	54
5.2	Arrays . . . . .	55
5.3	For Loops . . . . .	57
5.3.1	Nested For Loops . . . . .	60
5.4	While Loops . . . . .	61
5.5	Exercises . . . . .	61
<b>6</b>	<b>Creating Functions in Python</b>	<b>63</b>
6.1	Introduction . . . . .	63
6.2	Functions with multiple return values . . . . .	65
6.3	Exercises . . . . .	66
<b>7</b>	<b>Creating Classes in Python</b>	<b>69</b>
7.1	Introduction . . . . .	69
7.2	The <code>__init__()</code> Function . . . . .	70
7.3	Exercises . . . . .	73
<b>8</b>	<b>Creating Python Modules</b>	<b>74</b>
8.1	Python Modules . . . . .	74
8.2	Exercises . . . . .	75
<b>9</b>	<b>File Handling in Python</b>	<b>77</b>
9.1	Introduction . . . . .	77
9.2	Write Data to a File . . . . .	77
9.3	Read Data from a File . . . . .	78
9.4	Logging Data to File . . . . .	78
9.5	Web Resources . . . . .	79
9.6	Exercises . . . . .	79

<b>10 Error Handling in Python</b>	<b>82</b>
10.1 Introduction to Error Handling . . . . .	82
10.1.1 Syntax Errors . . . . .	82
10.1.2 Exceptions . . . . .	82
10.2 Exceptions Handling . . . . .	83
<b>11 Debugging in Python</b>	<b>85</b>
<b>12 Installing and using Python Packages</b>	<b>86</b>
12.1 What is PIP? . . . . .	86
<b>III Python Environments and Distributions</b>	<b>87</b>
<b>13 Introduction to Python Environments and Distributions</b>	<b>88</b>
13.1 Package and Environment Managers . . . . .	89
13.1.1 PIP . . . . .	89
13.1.2 Conda . . . . .	89
13.2 Python Virtual Environments . . . . .	90
<b>14 Anaconda</b>	<b>91</b>
14.1 Anaconda Navigator . . . . .	91
14.2 Anaconda Prompt . . . . .	91
<b>IV Python Editors</b>	<b>94</b>
<b>15 Python Editors</b>	<b>95</b>
<b>16 Spyder</b>	<b>97</b>
16.1 Configuration . . . . .	98
<b>17 Visual Studio Code</b>	<b>100</b>
17.1 Introduction to Visual Studio Code . . . . .	100
17.2 Python in Visual Studio Code . . . . .	101
<b>V Python for Mathematics Applications</b>	<b>102</b>
<b>18 Mathematics in Python</b>	<b>103</b>
18.1 Basic Math Functions . . . . .	103
18.1.1 Exercises . . . . .	105
18.2 Statistics . . . . .	107
18.2.1 Introduction to Statistics . . . . .	107
18.2.2 Statistics functions in Python . . . . .	108
18.3 Trigonometric Functions . . . . .	110
18.4 Polynomials . . . . .	114

<b>19 Linear Algebra in Python</b>	<b>117</b>
19.1 Introduction to Linear Algebra . . . . .	117
19.2 Linear Algebra with Python . . . . .	118
19.2.1 Vectors . . . . .	119
19.2.2 Matrices . . . . .	120
19.2.3 Linear Algebra (numpy.linalg) . . . . .	120
19.2.4 Matrix Addition . . . . .	120
19.2.5 Matrix Subtraction . . . . .	121
19.2.6 Matrix Multiplication . . . . .	122
19.2.7 Transpose of a Matrix . . . . .	125
19.2.8 Determinant . . . . .	126
19.2.9 Inverse Matrix . . . . .	127
19.3 Solving Linear Equations . . . . .	128
19.4 Exercises . . . . .	130
<b>20 Complex Numbers in Python</b>	<b>132</b>
20.1 Introduction to Complex Numbers . . . . .	132
20.2 Complex Numbers with Python . . . . .	134
<b>21 Differential Equations</b>	<b>136</b>
21.1 Introduction to Differential Equations . . . . .	136
21.2 ODE Solvers in Python . . . . .	139
21.3 Solving Multiple 1. order Differential Equations . . . . .	142
21.4 Solving Higher order Differential Equations . . . . .	145
21.5 Exercises . . . . .	147
<b>22 Optimization</b>	<b>152</b>
<b>VI Using Python for Simulations</b>	<b>156</b>
<b>23 Introduction to Simulations</b>	<b>157</b>
<b>24 Differential Equations</b>	<b>158</b>
24.1 Introduction to Differential Equations . . . . .	158
<b>25 Discrete Systems</b>	<b>160</b>
25.1 Discretization . . . . .	160
25.2 Exercises . . . . .	164
<b>26 Real-Time Simulations</b>	<b>166</b>
26.1 Introduction . . . . .	166
26.2 Introduction to Real-Time Plotting . . . . .	168
26.3 Real-Time Plotting with Animation . . . . .	173
26.3.1 Speeding Up the Plot Animation . . . . .	177
<b>VII Data Acquisition (DAQ) with Python</b>	<b>182</b>
<b>27 Plotting Sensor Data</b>	<b>183</b>
27.1 Introduction . . . . .	183

27.2	Introduction to Real-Time Plotting . . . . .	183
27.3	Real-Time Plotting with Animation . . . . .	185
27.3.1	Speeding Up the Plot Animation . . . . .	187
<b>28</b>	<b>Data Acquisition (DAQ) with Python</b>	<b>190</b>
28.1	Introduction to DAQ . . . . .	190
28.2	Data Acquisition using NI DAQ Devices . . . . .	190
28.2.1	NI-DAQmx . . . . .	192
28.2.2	Measurement Automation Explorer (MAX) . . . . .	193
28.3	NI-DAQmx Python API . . . . .	193
28.3.1	Analog Write . . . . .	194
28.3.2	Analog Read . . . . .	194
28.3.3	Digital Write . . . . .	196
28.3.4	Digital Read . . . . .	196
28.4	Controlling LEDs . . . . .	197
28.5	Read Data from Temperature Sensors . . . . .	199
28.5.1	Read Data from TMP36 Temperature Sensor . . . . .	199
28.5.2	Read Data from Thermistor . . . . .	203
28.5.3	Read Data NI TC-01 Thermocouple Device . . . . .	207
28.6	Data Logging . . . . .	208
<b>VIII</b>	<b>Control Systems</b>	<b>209</b>
<b>29</b>	<b>Python used for Control Applications</b>	<b>210</b>
29.1	Python Control Systems Library . . . . .	210
29.1.1	Python Control Systems Library Functions . . . . .	210
29.1.2	MATLAB compatibility module . . . . .	212
29.2	Control Systems . . . . .	212
29.2.1	Transfer functions . . . . .	213
29.2.2	State-space Models . . . . .	214
29.3	PID Control . . . . .	215
29.3.1	PI Control . . . . .	216
<b>30</b>	<b>Transfer Functions</b>	<b>218</b>
30.1	1.order Transfer Functions . . . . .	219
30.1.1	Step Response . . . . .	221
30.2	1.order Transfer Functions with Time Delay . . . . .	224
30.3	Integrator Transfer Functions . . . . .	226
30.4	2.order Transfer Functions . . . . .	229
30.5	Block Diagrams . . . . .	230
30.5.1	Serial Block Diagrams . . . . .	230
30.5.2	Parallel Block Diagrams . . . . .	231
30.5.3	Feedback Block Diagrams . . . . .	233
<b>31</b>	<b>State Space Models</b>	<b>235</b>
31.1	Introduction . . . . .	235
31.2	Discrete State-space Models . . . . .	238

<b>32 Frequency Response</b>	<b>239</b>
32.1 Introduction . . . . .	239
32.1.1 Theory . . . . .	239
32.2 Bode Diagram . . . . .	240
<b>33 Stability Analysis</b>	<b>246</b>
33.1 Introduction . . . . .	246
33.1.1 Asymptotically Stable System . . . . .	246
33.1.2 Marginally Stable System . . . . .	246
33.1.3 Unstable System . . . . .	247
33.2 Poles . . . . .	249
33.2.1 Asymptotically Stable System . . . . .	249
33.2.2 Marginally Stable System . . . . .	249
33.2.3 Unstable System . . . . .	250
33.3 Feedback Systems . . . . .	250
33.3.1 Loop Transfer Function . . . . .	250
33.3.2 Tracking Transfer Function . . . . .	251
33.3.3 Sensitivity Transfer Function . . . . .	251
33.3.4 Characteristic Polynomial . . . . .	252
33.4 Frequency Response and stability Analysis . . . . .	253
33.5 Examples . . . . .	256
<b>34 Air Heater Control System</b>	<b>269</b>
34.1 Introduction . . . . .	269
34.2 Discrete Air Heater . . . . .	271
34.3 Transfer Function . . . . .	275
34.4 Stability Analysis . . . . .	278
34.5 Ziegler–Nichols Frequency Response Method . . . . .	285
34.6 Air Heater Control System . . . . .	288
<b>IX Python Database Development</b>	<b>294</b>
<b>35 Database Applications with Python</b>	<b>295</b>
35.1 Structured Query Language (SQL) . . . . .	295
35.2 SQL Server . . . . .	296
35.3 MySQL . . . . .	296
35.4 MongoDB . . . . .	296
<b>36 SQL Server with Python</b>	<b>297</b>
36.1 Introduction to SQL Server . . . . .	297
36.2 SQL Server drivers for Python . . . . .	297
36.3 pyodbc . . . . .	297
36.3.1 Installation of pyodbc . . . . .	297
36.3.2 ODBC Drivers . . . . .	297
36.4 SQL Server Python Examples . . . . .	298
36.5 Stored Procedures . . . . .	299
36.6 Resources . . . . .	299
36.7 pymssql . . . . .	299
36.8 Resources . . . . .	299

<b>X Python Application Development</b>	<b>300</b>
<b>37 OPC Communication with Python</b>	<b>301</b>
37.1 Introduction to OPC . . . . .	301
37.2 OPC Classic . . . . .	302
37.3 OPC UA . . . . .	303
37.4 OPC Examples with Python . . . . .	304
<b>38 Python Integration with LabVIEW</b>	<b>305</b>
38.1 What is LabVIEW? . . . . .	305
38.2 Using Python in LabVIEW . . . . .	305
<b>39 Raspberry Pi and Python</b>	<b>310</b>
39.1 What is Raspberry Pi? . . . . .	310
<b>XI Resources</b>	<b>311</b>
<b>40 Python for MATLAB Users</b>	<b>312</b>
40.1 Use Python inside MATLAB . . . . .	313
40.2 Calling MATLAB from Python . . . . .	314
<b>41 Python Resources</b>	<b>316</b>
41.1 Python Distributions . . . . .	316
41.2 Python Libraries . . . . .	316
41.3 Python Editors . . . . .	316
41.4 Python Tutorials . . . . .	317
41.5 Python in Visual Studio . . . . .	317
<b>XII Solutions to Exercises</b>	<b>319</b>

# **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 is 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].



## 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, eg., the Django Web framework), etc.

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.

Resources:

- [The official Python Tutorial \(3.7\)](#)
- [The official Python Documentation \(3.7\)](#)
- [Python Tutorial \(w3schools.com\)](#)

## 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:

- [Spyder](#)
- [Visual Studio Code](#)
- [Visual Studio](#)

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

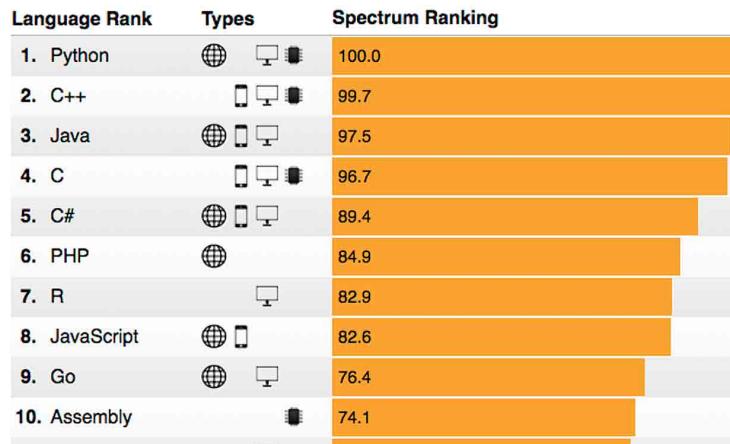


Figure 1.2: The Most Popular Programming Languages

- 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.

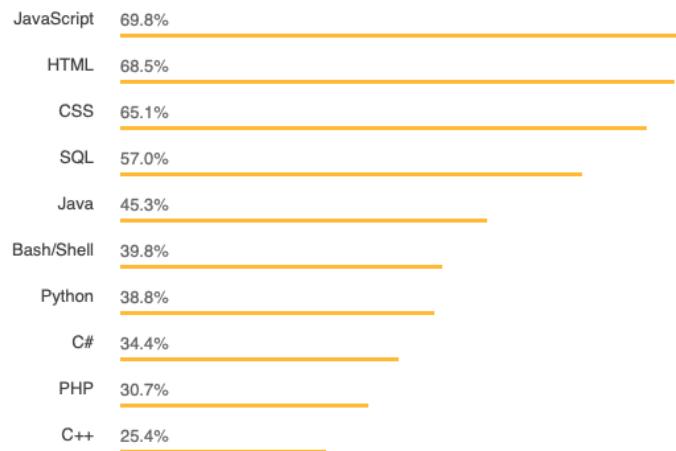


Figure 1.3: The Top Programming Languages - Stack Overflow Survey

- 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)

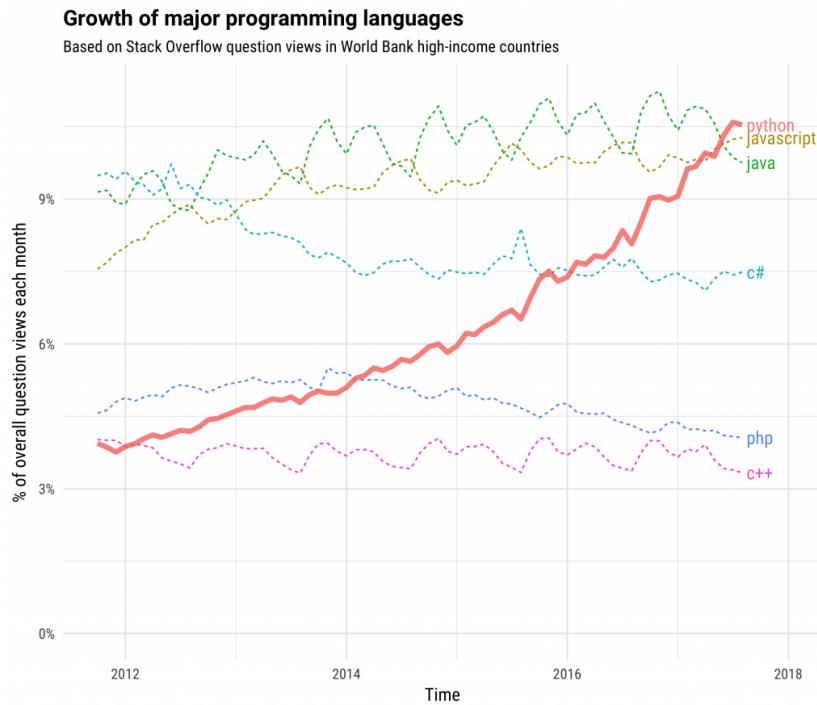


Figure 1.4: The Incredible Growth of Python

- 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/>

### Projections of future traffic for major programming languages

Future traffic is predicted with an STL model, along with an 80% prediction interval.

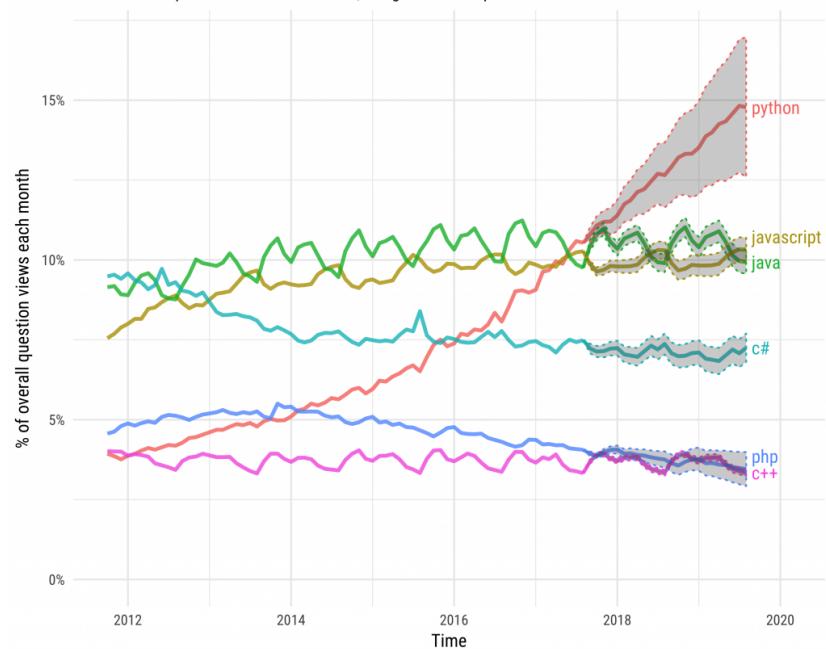


Figure 1.5: The Future of Python

# 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. **C<sub>P</sub>ython** is the reference implementation of the Python programming language. Written in C, C<sub>P</sub>ython 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



The screenshot shows a window titled "Python 3.7.0 Shell". The shell displays the following text:  
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!  
>>>  
In the bottom right corner of the shell window, there is a status bar with the text "Ln: 6 Col: 4".

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.

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\\_\(Python\\_distribution\)](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
- 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 you 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 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\\_\(software\)](https://en.wikipedia.org/wiki/Spyder_(software))

Spyder is included in the Anaconda Distribution.

#### **2.4.4 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 default (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](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio)

#### **2.4.5 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/>

#### 2.4.6 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

#### 2.4.7 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](https://en.wikipedia.org/wiki/Project_Jupyter)

### 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

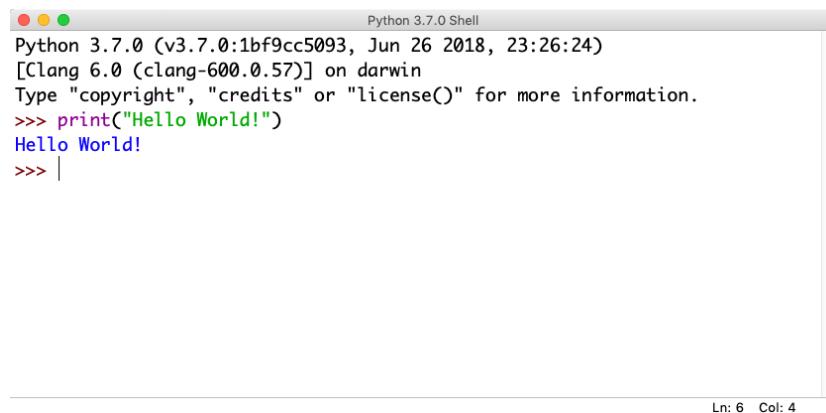
## Start using Python

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.



A screenshot of the Python 3.7.0 Shell window. The title bar reads "Python 3.7.0 Shell". The main area displays the following text:  
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.

You 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.

### 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.

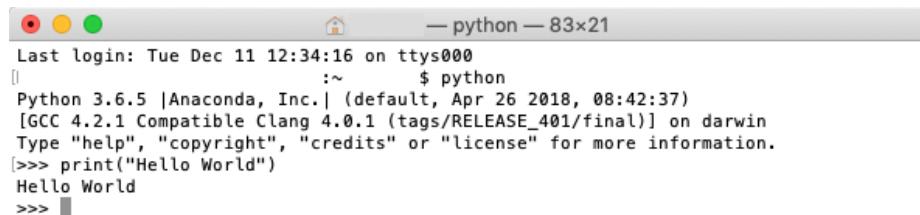
```
1 Last login: Tue Dec 11 12:34:16 on ttys000  
2 Hans-Petter-Work-MacBook-Air:~ hansha$ python  
3 Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)  
4 [GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on  
darwin  
5 Type "help", "copyright", "credits" or "license" for more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

The prompt `>>>` on the last line indicates that you are now in an interactive Python interpreter 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.



A screenshot of a macOS Terminal window titled “python — 83x21”. The window shows the following text:

```
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
```

### 3.4.2 Opening the Console on Windows

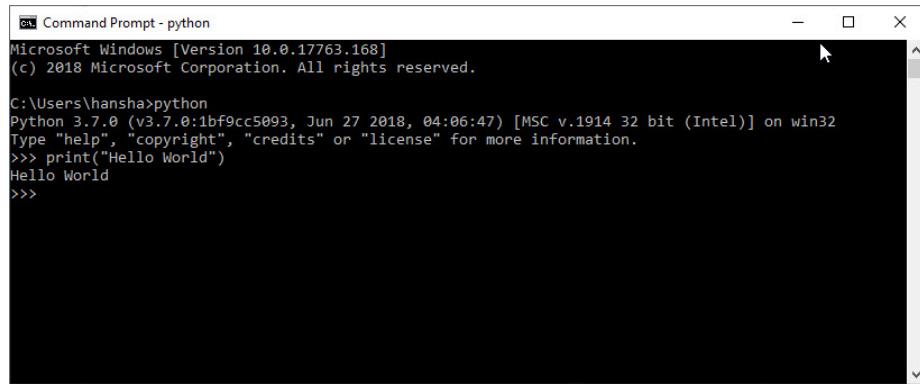
Window's console is called the Command Prompt, named cmd. An easy way to get to it is by using the key combination Windows+R (Windows meaning the windows logo button), which should open a Run dialog. Then type cmd and hit Enter or click Ok.

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.



```
Command Prompt - python
Microsoft Windows [Version 10.0.17763.168]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\hansha>python
Python 3.7.0 (v3.7.0:bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

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.

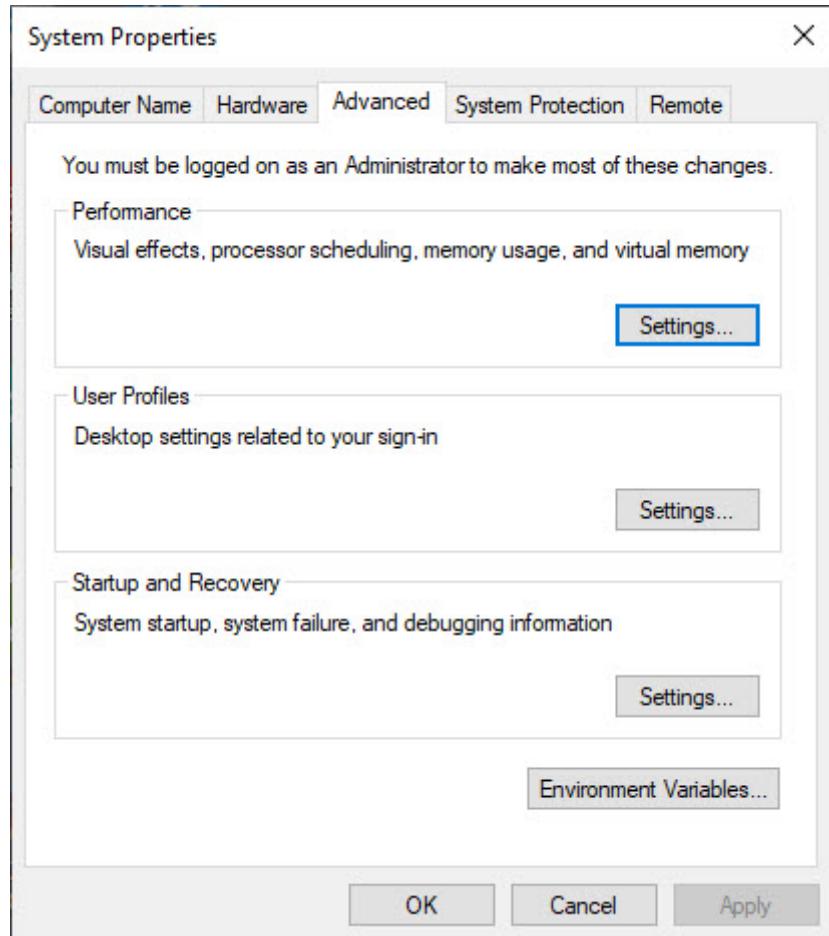


Figure 3.4: Windows System Properties

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.

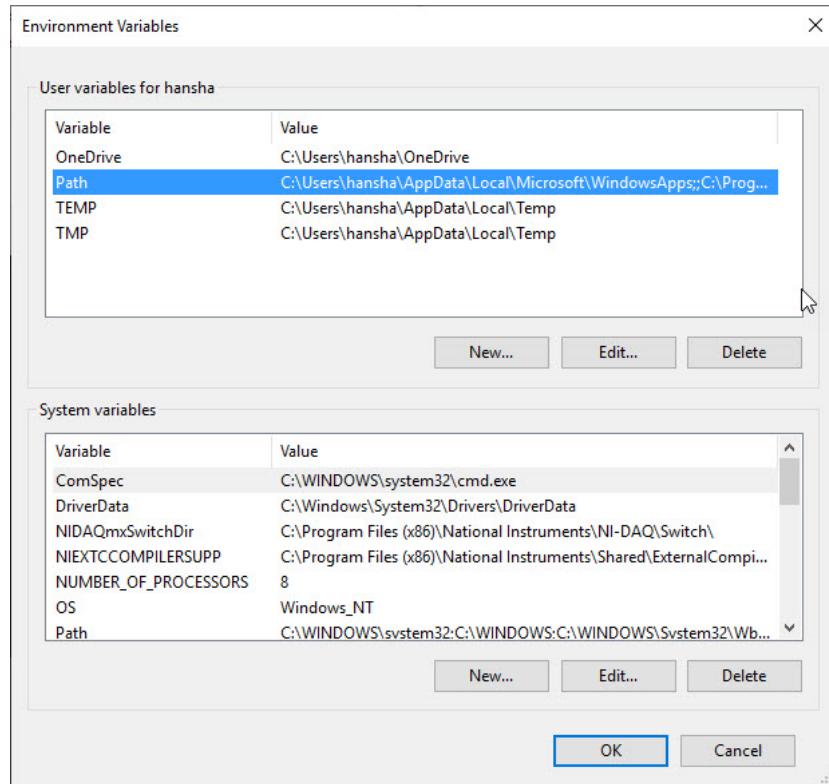


Figure 3.5: Windows System Properties

## 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 Python 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.

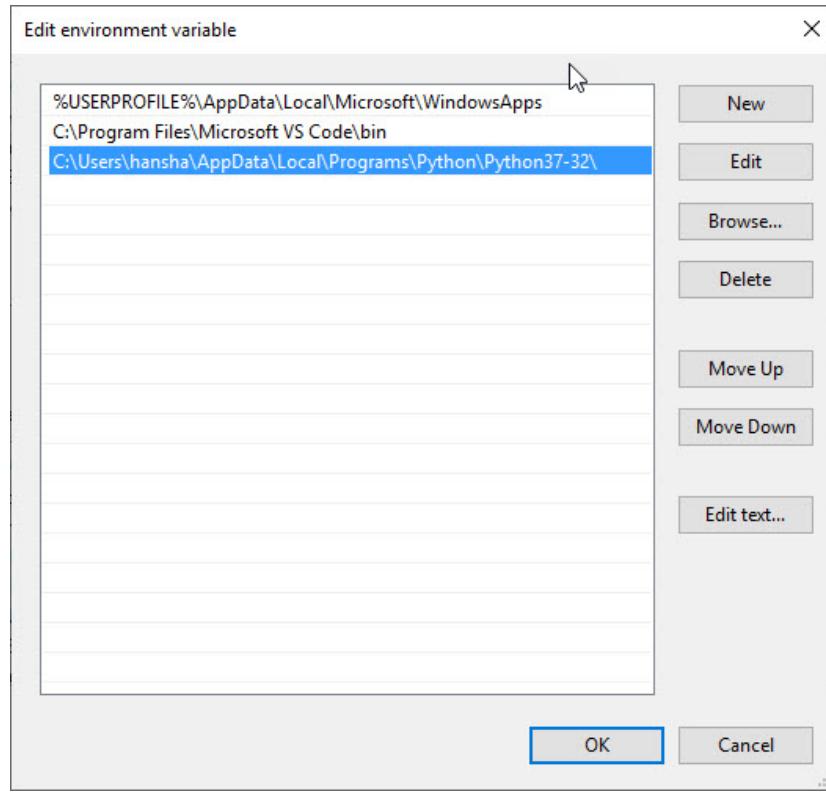


Figure 3.6: Windows System Properties

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:

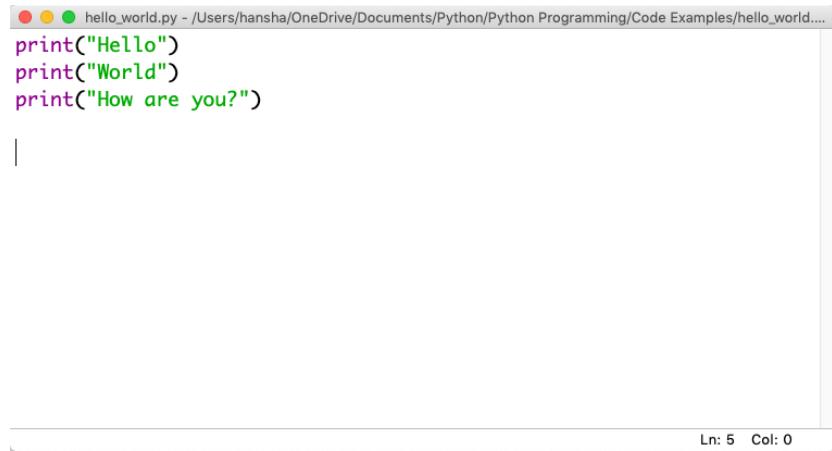
```
1 $ cd /Users/username/Downloads  
2 $ python helloworld.py
```

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?
```



```
● ○ ● hello_world.py - /Users/hansha/OneDrive/Documents/Python/Python Programming/Code Examples/hello_world....  
print("Hello")  
print("World")  
print("How are you?")
```

Ln: 5 Col: 0

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.

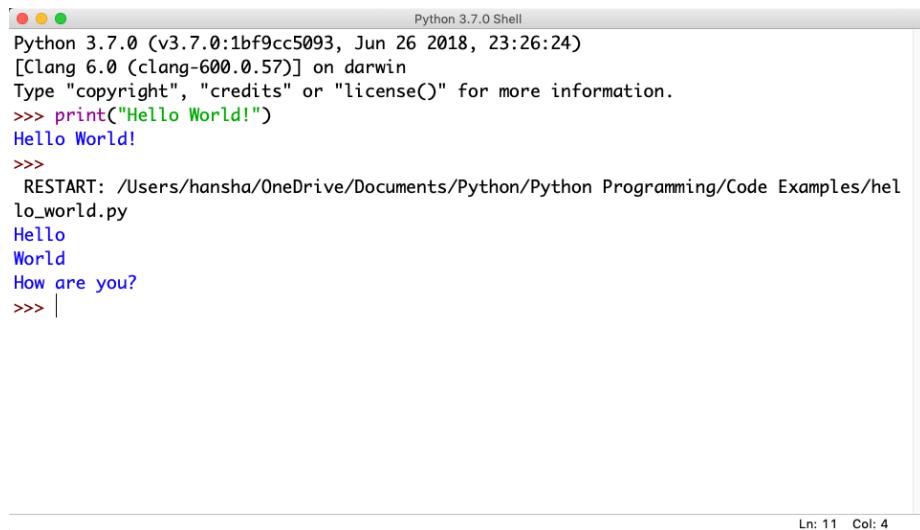
Then it responds with:

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

### 3.5.4 Run Python Scripts from Spyder

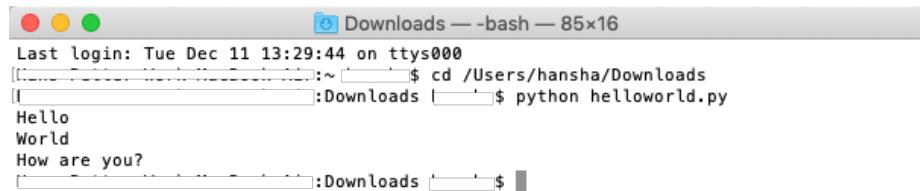
If you have installed the Anaconda distribution package you can use the Spyder editor. See 3.11.

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.



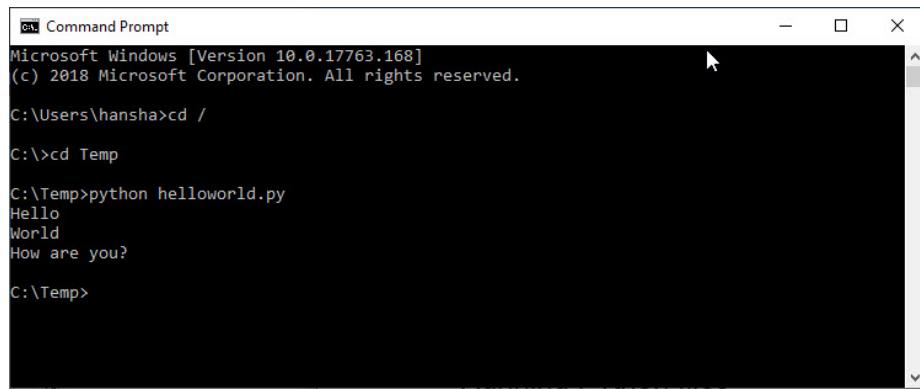
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)  
[Clang 6.0 (clang-600.0.57)] on darwin  
Type "copyright", "credits" or "license()" for more information.  
>>> print("Hello World!")  
Hello World!  
>>>  
RESTART: /Users/hansha/OneDrive/Documents/Python/Python Programming/Code Examples/hello\_world.py  
Hello  
World  
How are you?  
>>> |

Figure 3.8: Running a Python Script



Downloads — bash — 85x16  
Last login: Tue Dec 11 13:29:44 on ttys000  
[User] ~ % cd /Users/hansha/Downloads  
[User] Downloads % python helloworld.py  
Hello  
World  
How are you?  
[User] Downloads %

Figure 3.9: Running Python Scripts from Console window on macOS



Command Prompt  
Microsoft Windows [Version 10.0.17763.168]  
(c) 2018 Microsoft Corporation. All rights reserved.  
C:\Users\hansha>cd /  
C:\>cd Temp  
C:\Temp>python helloworld.py  
Hello  
World  
How are you?  
C:\Temp>

Figure 3.10: Running Python Scripts from Console window on macOS

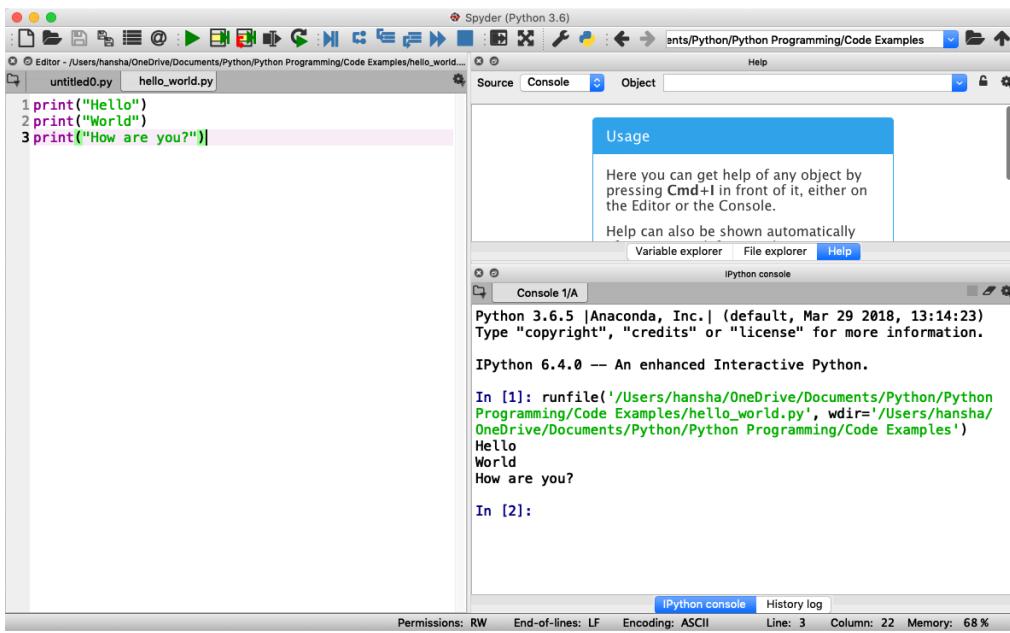


Figure 3.11: Running a Python Script in Spyder

# 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.

### **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

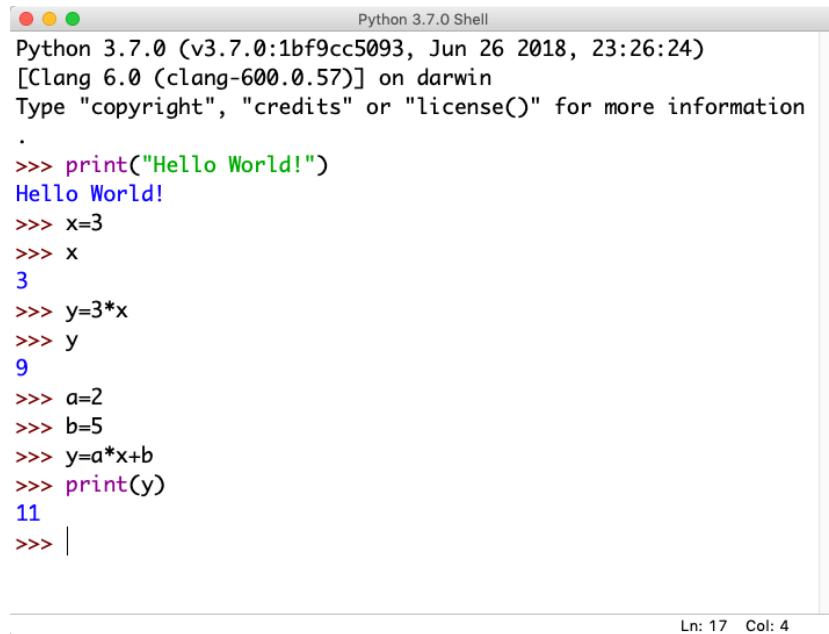
As seen in the examples, you can use the *print()* command in order to show the values on the screen.

[End of Example]

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.



The screenshot shows a window titled "Python 3.7.0 Shell". The console output is as follows:

```
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!
>>> x=3
>>> x
3
>>> y=3*x
>>> y
9
>>> a=2
>>> b=5
>>> y=a*x+b
>>> print(y)
11
>>> |
```

Ln: 17 Col: 4

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

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):

Name	Type	Size	Value
x	int	1	1
y	float	1	2.8
z	complex	1	(3+2j)

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())
```

```
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 for 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 import 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:

```
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/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

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
```

```
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.

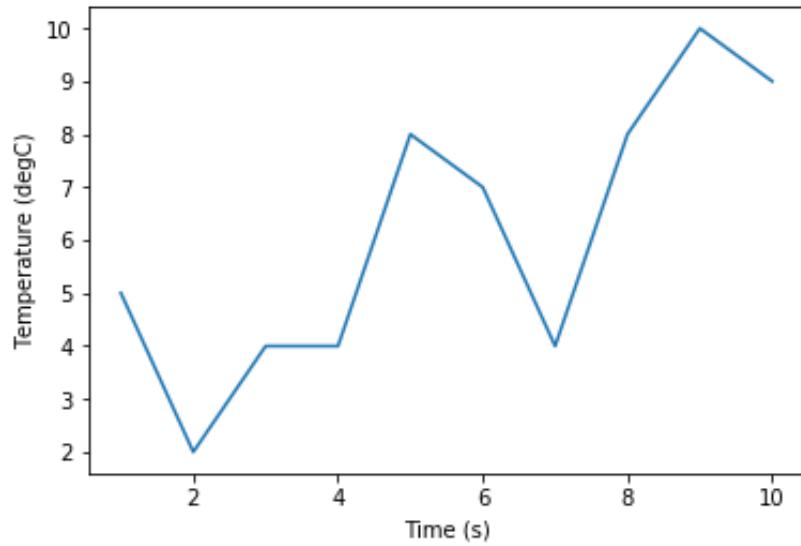


Figure 4.3: Plotting in Python

[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:

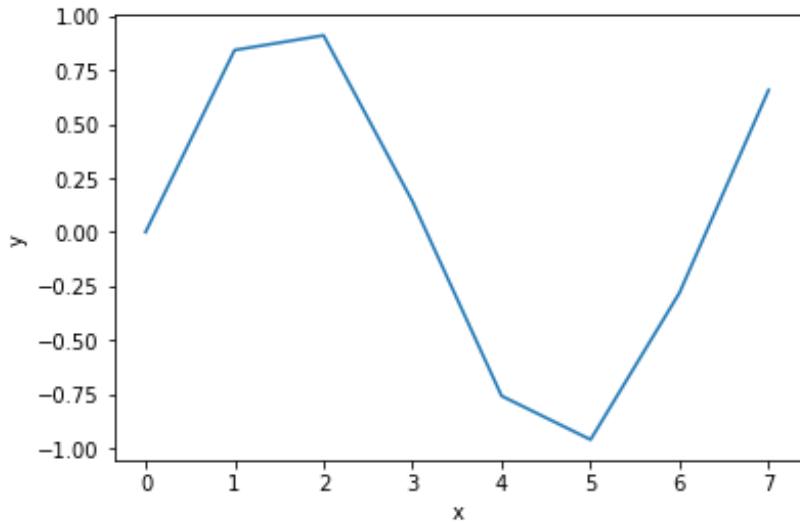


Figure 4.4: Plotting a Sine function in Python

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 xstart = 0
5 xstop = 2*np.pi
6 increment = 0.1
7
8 x = np.arange(xstart,xstop,increment)
9
10 y = np.sin(x)
11
12 plt.plot(x, y)
13 plt.xlabel('x')
14 plt.ylabel('y')
15 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

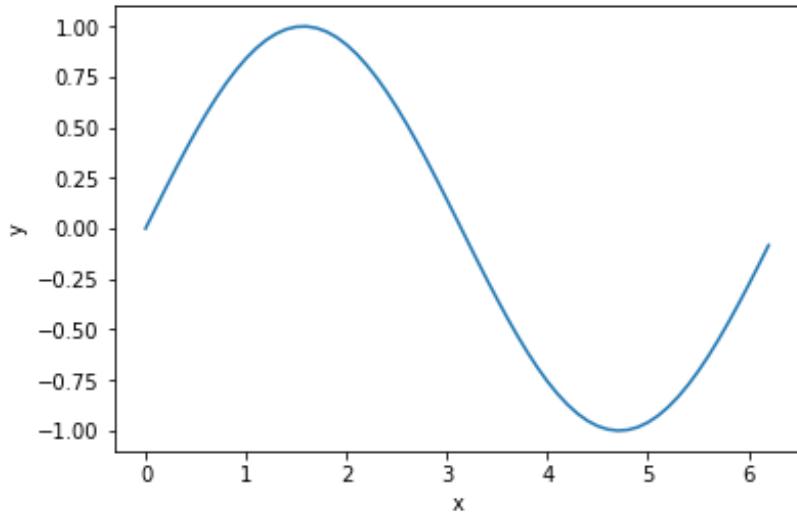


Figure 4.5: Plotting a Sine function in Python - Better Implementation

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

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 xstart = 0
5 xstop = 2*np.pi
6 increment = 0.1
7
8 x = np.arange(xstart,xstop,increment)
9
10 y = np.sin(x)
11 z = np.cos(x)
12
13
14
15 plt.subplot(2,1,1)
16 plt.plot(x, y, 'g')
17 plt.title('sin')
18 plt.xlabel('x')
19 plt.ylabel('sin(x)')
20 plt.grid()
21 plt.show()
22
23
24 plt.subplot(2,1,2)
25 plt.plot(x, z, 'r')
26 plt.title('cos')
27 plt.xlabel('x')
28 plt.ylabel('cos(x)')
29 plt.grid()
30 plt.show()

```

[End of Example]

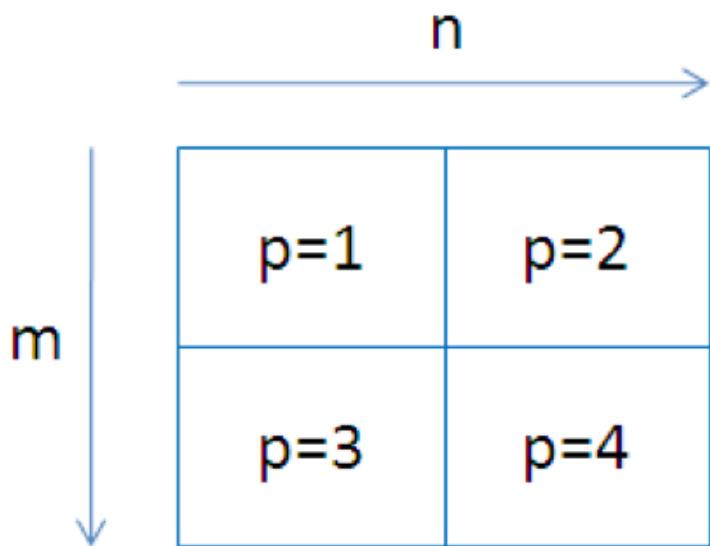


Figure 4.6: Creating Subplots in Python

#### 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 things, 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:
```

```
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 N = len(data)
3
4 print(N)
5
6 print(data[2])
7
8 data[2] = 7.3
9
10 print(data[2])
```

```
12
13
14 for x in data:
15     print(x)
16
17
18 data.append(11.4)
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 carlist = ["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(11.4)
```

[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.

```
1 data = [1, 5, 6, 3, 12, 3]
2
3 sum = 0
4
5 #Find the Sum of all the numbers
6 for x in data:
7     sum = sum + x
8
9 print(sum)
10
11 #Find the Mean or Average of all the numbers
12
13 N = len(data)
14
15 mean = sum/N
16
17 print(mean)
```

This gives the following results:

```
1 30
2 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.

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

$$f_n = f_{n-1} + f_{n-2} \quad (5.1)$$

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

```

```

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

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:

$$\dot{x} = ax \quad (5.2)$$

Where:

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

where T is the time constant.

The solution for the differential equation is:

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

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:

$$0 \leq t \leq 25$$

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 indentation 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

[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:

```
1 def average(x,y):  
2     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:

```
1 from myfunctions import average  
2  
3 a = 2  
4 b = 3  
5  
6 c = average(a,b)  
7  
8 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:

```
1 def stat(x):  
2     totalsum = 0  
3  
4     #Find the Sum of all the numbers  
5     for x in data:  
6         totalsum = totalsum + x  
7  
8  
9     #Find the Mean or Average of all the numbers  
10    N = len(data)  
11  
12    mean = totalsum/N  
13  
14  
15    return totalsum, mean  
16  
17  
18  
19
```

```

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

```

Listing 6.5: Function with multiple return values

[End of Example]

### 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.

[End of Exercise]

**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:

$$2\pi[\text{radians}] = 360[\text{degrees}] \quad (6.1)$$

This gives:

$$d[\text{degrees}] = r[\text{radians}] \times \left(\frac{180}{\pi}\right) \quad (6.2)$$

and

$$r[\text{radians}] = d[\text{degrees}] \times \left(\frac{\pi}{180}\right) \quad (6.3)$$

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.

[End of Exercise]

**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  $F_n$  of Fibonacci numbers is defined by the recurrence relation

$$f_n = f_{n-1} + f_{n-2} \quad (6.4)$$

with seed values:

$$f_0 = 0, f_1 = 1$$

Create a Function that Implementing the N first Fibonacci Numbers

[End of Exercise]

**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**:

```
1 class ClassName:  
2     <statement-1>  
3  
4     .  
5  
6     .  
7     .  
8     <statement-N>
```

#### Example 7.1.1. Simple Class Example

We will create a simple Class in Python.

```
1 class Car:  
2     model = "Volvo"  
3     color = "Blue"  
4  
5 car = Car()  
6  
7  
8 print(car.model)  
9 print(car.color)
```

Listing 7.1: Simple Python Class

The results will be in this case:

```
1 Volvo  
2 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)
```

```

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:

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

[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 7.5: 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 7.6: Script that is using the Class

Notice the following line at the top:

```
1 from Car import Car
```

[language=Python]

[End of Example]

## 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:

$$T_f = (T_c \times 9/5) + 32 \quad (7.1)$$

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \quad (7.2)$$

[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:

$$T_f = (T_c \times 9/5) + 32 \quad (8.1)$$

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \quad (8.2)$$

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

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

Listing 8.1: Fahrenheit Functions

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))
```

Listing 8.2: Python Script testing the functions

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

and degrees.

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

$$2\pi[\text{radians}] = 360[\text{degrees}] \quad (8.3)$$

This gives:

$$d[\text{degrees}] = r[\text{radians}] \times \left(\frac{180}{\pi}\right) \quad (8.4)$$

and

$$r[\text{radians}] = d[\text{degrees}] \times \left(\frac{\pi}{180}\right) \quad (8.5)$$

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](https://www.w3schools.com/python/python_file_handling.asp)

Reading and Writing Files - python.org:

<https://docs.python.org/3/tutorial/inputoutput.html#reading-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.

[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:

```
1 import random
2 for x in range(10):
3     data = random.randint(1,31)
4     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:

```
1 >>> print(Hello World)
2     File "<ipython-input-1-10cb182148e3>", line 1
3         print(Hello World)
4             ^
5 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:

```
1 >>> 10 * (1/0)
2 Traceback (most recent call last):
3
4   File "<ipython-input-2-0b280f36835c>", line 1, in <module>
5       10 * (1/0)
6
7 ZeroDivisionError: division by zero
```

or:

```
1 >>> '2' + 2
2 Traceback (most recent call last):
3
```

```
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.

# **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](http://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 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\\_\(Python\\_distribution\)](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
```

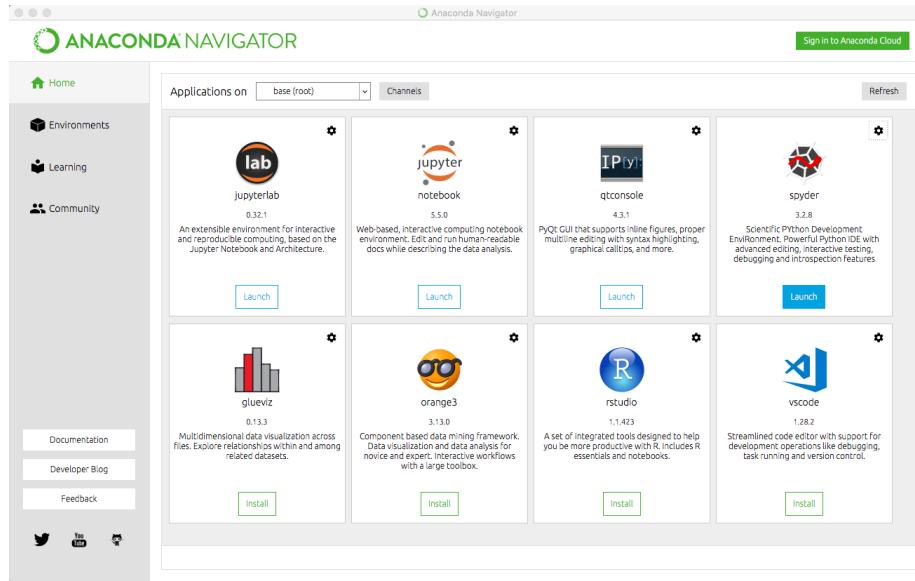


Figure 14.1: Anaconda Navigator

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

For 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.

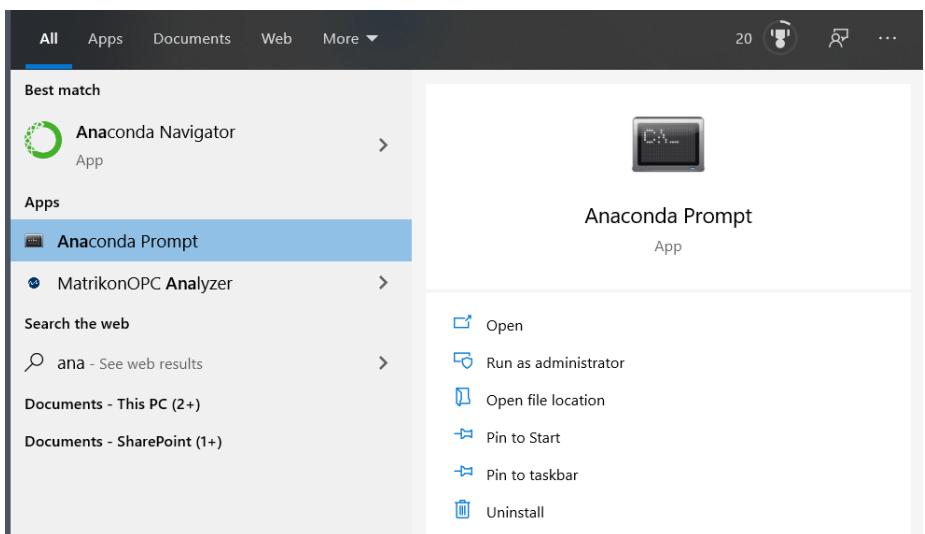


Figure 14.2: Anaconda Navigator

## **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
- 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 you are going 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 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.

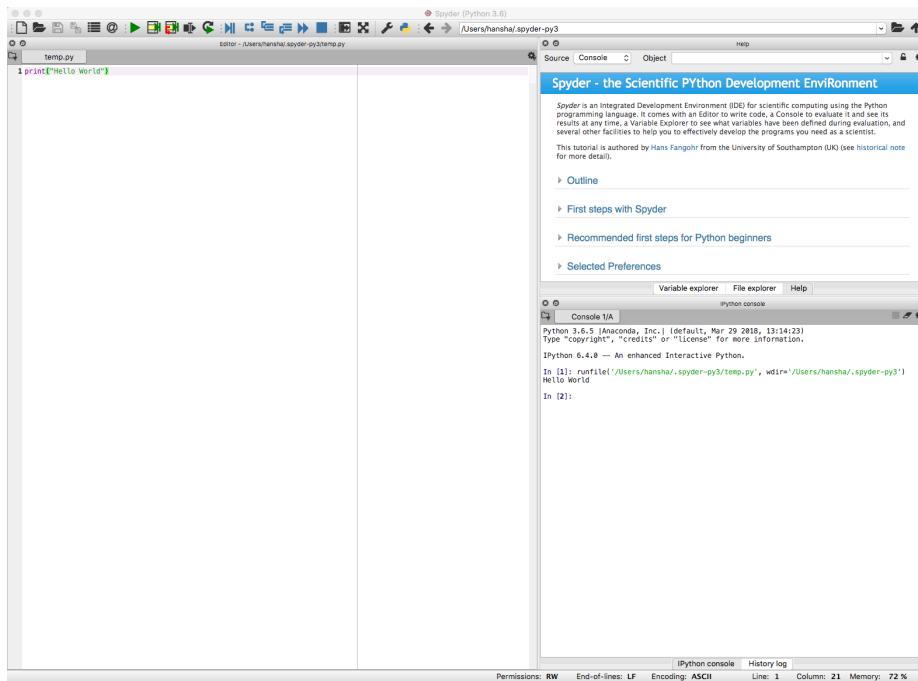


Figure 16.1: Spyder Editor

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.

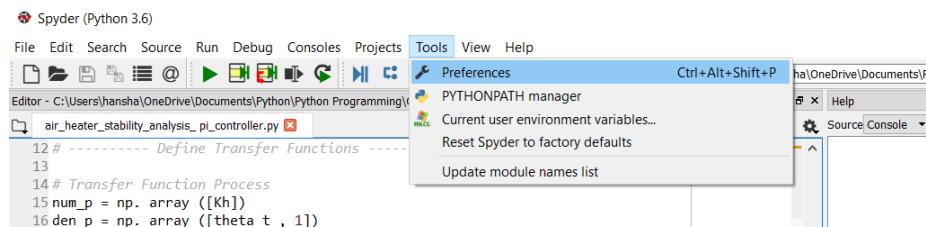


Figure 16.2: Python Tools-Preferences

Then select "Automatic" as shown in Figure 16.3.

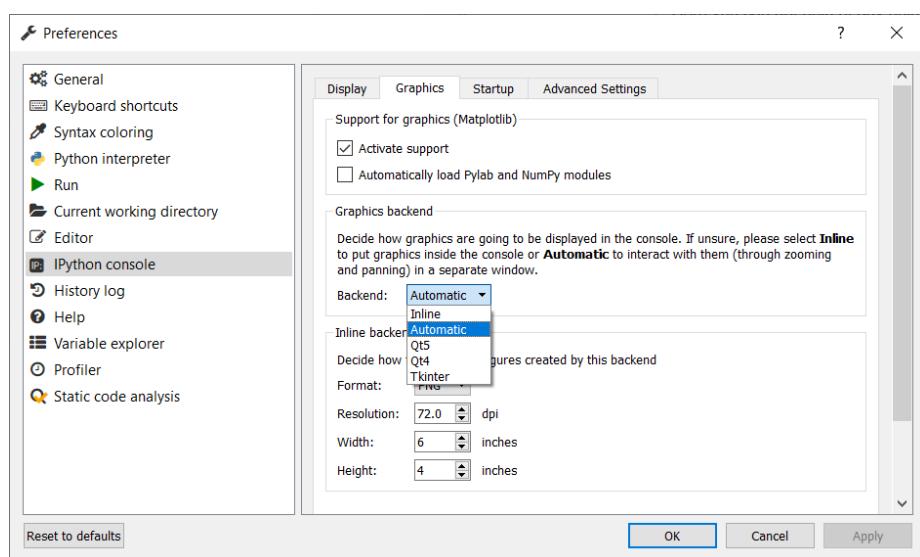


Figure 16.3: Python Preferences window

# 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.

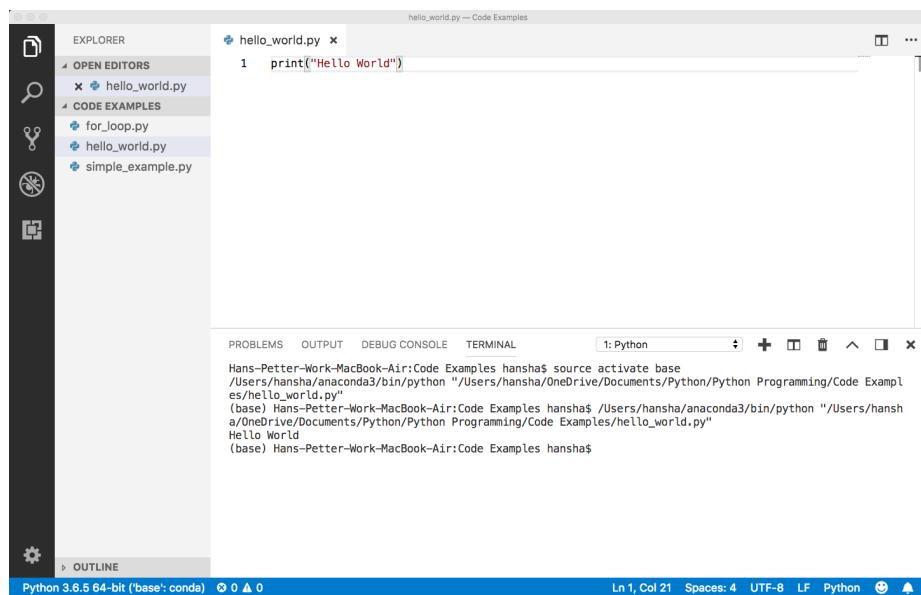


Figure 17.1: Using Visual Studio Code as Python Editor

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

Web:

<https://code.visualstudio.com>

Wikipedia:

[https://en.wikipedia.org/wiki/Visual\\_Studio\\_Code](https://en.wikipedia.org/wiki/Visual_Studio_Code)

## 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).

<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 import 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:

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

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:

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

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 = ...
den = ...
f = ...
```

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

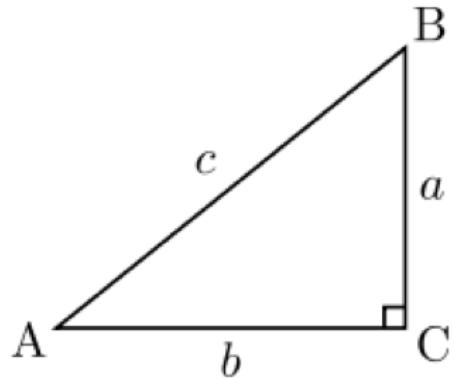


Figure 18.1: Right-angled triangle

Pythagoras theorem is as follows:

$$c^2 = a^2 + b^2 \quad (18.3)$$

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:

$$E = mc^2 \quad (18.4)$$

The sun radiates  $385 \times 10^{24} J/s$  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  $2 \times 10^{30} kg$ .

[End of Exercise]

**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 ??.

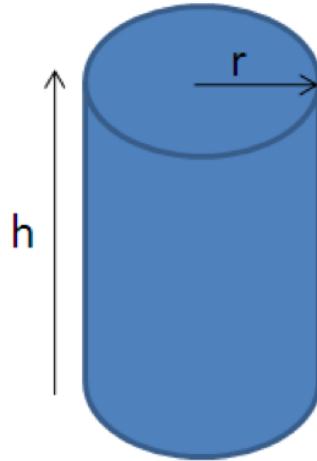


Figure 18.2: cylinder

[End of Exercise]

## 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:

$$\bar{x} = \frac{x_1 + x_2 + x_3 + \dots + x_N}{N} = \frac{1}{N} \sum_{i=1}^N x_i \quad (18.5)$$

#### Example 18.2.1. Mean

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

Mean:

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

[End of Example]

### Variance:

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

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

### 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.

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

We typically use the symbol  $\sigma$  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:

```
1 import statistics as st
2
3 data = [-1.0, 2.5, 3.25, 5.75]
4
5 #Mean or Average
6 m = st.mean(data)
7 print(m)
8
9 # Standard Deviation
10 st.dev = st.stdev(data)
```

```

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 # Maximum Value
22 maxv = np.max(data)
23 print(maxv)

```

Listing 18.2: Statistics using the NumPy Library

[End of Example]

**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

```
1 import math as mt
2
3 x = 2*mt.pi
4
5 y = mt.sin(x)
6 print(y)
7
8 y = mt.cos(x)
9 print(y)
10
11 y = mt.tan(x)
12 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.

```
1 import math as mt
2 import matplotlib.pyplot as plt
3
4 xdata = []
5 ydata = []
6
7 for x in range(0, 10):
8     xdata.append(x)
9     y = mt.sin(x)
10    ydata.append(y)
11
12 plt.plot(xdata, ydata)
13 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.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 xstart = 0
5 xstop = 2*np.pi
6 increment = 0.1
7
8 x = np.arange(xstart, xstop, increment)
9
10 y = np.sin(x)
```

```

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 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:

$$2\pi[\text{radians}] = 360[\text{degrees}] \quad (18.9)$$

This gives:

$$d[\text{degrees}] = r[\text{radians}] \times \left(\frac{180}{\pi}\right) \quad (18.10)$$

and

$$r[\text{radians}] = d[\text{degrees}] \times \left(\frac{\pi}{180}\right) \quad (18.11)$$

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.

[End of Exercise]

### 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:

$$\sin(A) = \frac{a}{c} \rightarrow A = \arcsin\left(\frac{a}{c}\right) \quad (18.12)$$

$$\cos(A) = \frac{b}{c} \rightarrow A = \arccos\left(\frac{b}{c}\right) \quad (18.13)$$

$$\tan(A) = \frac{a}{b} \rightarrow A = \arctan\left(\frac{a}{b}\right) \quad (18.14)$$

We may also need to use the Pythagoras' theorem:

$$c^2 = a^2 + b^2 \quad (18.15)$$

```
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.

[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**.

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

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  $0 \leq \theta \leq 2\pi$  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:

$$p(x) = p_1x^n + p_2x^{n-1} + \dots + p_nx + p_{n+1} \quad (18.17)$$

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

We will use the Polynomial Module in the NumPy Package.

Web:

<https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.polynomials.polynomial.html>

Other Resources:

Python Advanced Course Topics - Polynomials:

[https://www.python-course.eu/polynomiaclass\\_in\\_python.php](https://www.python-course.eu/polynomiaclass_in_python.php)

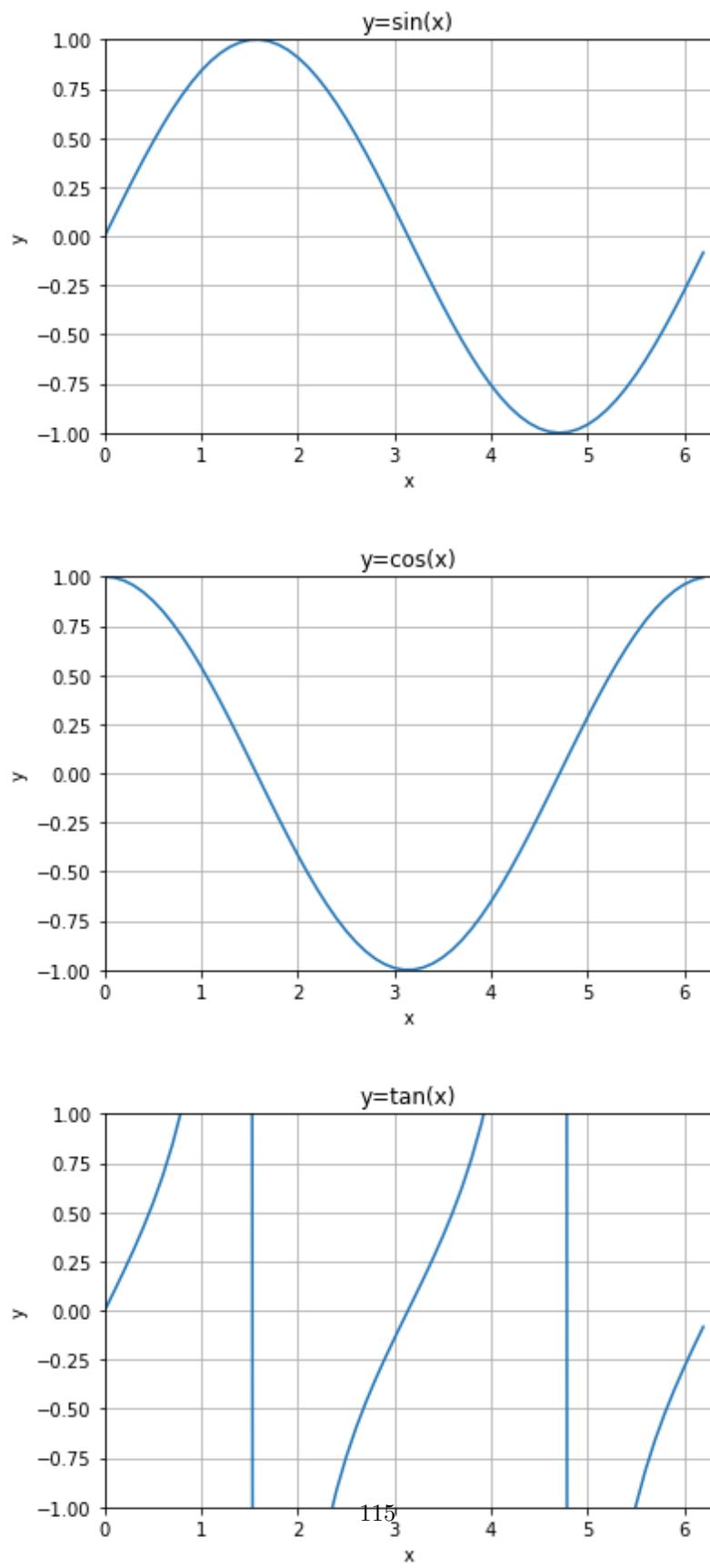


Figure 18.3: Trigonometric Functions

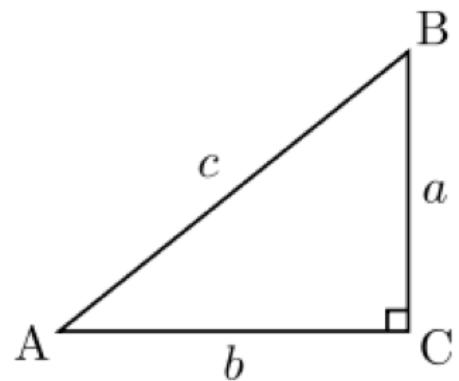


Figure 18.4: Right Triangle

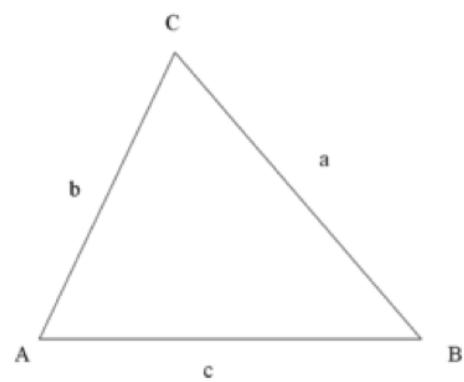


Figure 18.5: Law of Cosines

# 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:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \in R^{n \times m} \quad (19.1)$$

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

Example of a 3 x 3 matrix:

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

Example of a 3 x 4 matrix:

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

Example of a 4 x 2 matrix:

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

Python doesn't have a built-in type for matrices. However, we can treat 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:

```

1 a = [1, 3, 7, 2]
2
3 print("a =", a)
4
5
6 A = [[1, 3, 7, 2],
7     [5, 8, -9, 0],
8     [6, -7, 11, 12]]
9
10 print("A =", A)

```

Listing 19.1: Python Arrays

This gives the following output:

```

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

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 print("A =", A)
14
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 it is even better to 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)
```

### 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:

$$A \in R^{n \times m}$$

and

$$B \in R^{n \times m}$$

Then

$$C = A + B \in R^{n \times m}$$

Example:

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

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

Then we get:

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

#### **Example 19.2.2.** Matrix Addition in Python

```

1 import numpy as np
2
3 A = np.matrix([[0, 1],
4                 [-2, -3]])
5
6 B = np.matrix([[1, 0],
7                 [3, -2]])
8
9 C = A + B
10 print(C)

```

Listing 19.4: Matrix Addition in Python

We get:

```

1 [[ 1  1]
2  [ 1 -5]]

```

[End of Example]

#### **19.2.5 Matrix Subtraction**

Given the matrices:

$$A \in R^{n \times m}$$

and

$$B \in R^{n \times m}$$

Then

$$C = A - B \in R^{n \times m}$$

Example:

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

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

Then we get:

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

#### Example 19.2.3. Matrix Subtraction in Python

```

1 import numpy as np
2
3 A = np.matrix([[0, 1],
4                 [-2, -3]])
5
6 B = np.matrix([[1, 0],
7                 [3, -2]])
8
9 C = A - B
10 print(C)

```

Listing 19.5: Matrix Subtraction in Python

We get:

```

1 [[-1  1]
2  [-5 -1]]

```

[End of Example]

#### 19.2.6 Matrix Multiplication

Given the matrices:

$$A \in R^{n \times m}$$

and

$$B \in R^{m \times p}$$

Then

$$C = AB \in R^{n \times p}$$

Where

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

Example:

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

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

Then we get:

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

We do the same in Python:

#### Example 19.2.4. Matrix Multiplication in Python

```
1 import numpy as np
2
3 A = np.matrix([[0, 1],
4                 [-2, -3]])
5
6 B = np.matrix([[1, 0],
7                 [3, -2]])
8
9 C = A * B
10 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.

$$n \begin{bmatrix} m \\ A \end{bmatrix} m \begin{bmatrix} p \\ B \end{bmatrix} = n \begin{bmatrix} p \\ C \end{bmatrix}$$

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:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \in R^{n \times m} \quad (19.18)$$

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

The transpose of matrix a is then:

$$A^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ a_{12} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{1m} & a_{n2} & \dots & a_{nm} \end{bmatrix} \in R^{m \times n} \quad (19.19)$$

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

```

1 import numpy as np
2
3 A = np.matrix([[0, 1],
4                 [-2, -3]])
5
6 At = np.transpose(A)
7 print(At)
8
9
10 B = np.matrix([[1, 0, 4],
11                  [3, -2, 8]])
12
13 Bt = np.transpose(B)
14 print(Bt)
15
16
17 C = np.matrix([[1, 4],
18                  [2, -3],
19                  [-6, -2]])
20
21 Ct = np.transpose(C)
22 print(Ct)
23

```

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:

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

We have:

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

Example:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad (19.22)$$

We get:

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

#### Example 19.2.6. Determinant

Python Example:

```

1 import numpy as np
2 import numpy.linalg as la
3
4 A = np.matrix([[1, 2],
5                 [3, 4]])
6
7 Adet = la.det(A)
8
9 print(Adet)

```

Listing 19.9: Determinant

This gives:

```
1 -2.0000000000000004
```

Listing 19.10: Determinant

[End of Example]

### 19.2.9 Inverse Matrix

The inverse of a quadratic matrix  $A \in R^{n \times n}$  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:

$$AA^{-1} = A^{-1}A = I \quad (19.24)$$

For a  $2 \times 2$  matrix we have:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad (19.25)$$

The inverse of A becomes:

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \quad (19.26)$$

#### Example 19.2.7. Inverse Matrix

Python Example:

```
1 import numpy as np
2 import numpy.linalg as la
3
4 A = np.matrix([[1, 2],
5                 [3, 4]])
6
7 Ainv = la.inv(A)
8
9 print(Ainv)
```

Listing 19.11: Inverse Matrix

We get the following results:

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

[End of Example]

### 19.3 Solving Linear Equations

**Example 19.3.1.** Solving Linear Equations

Given the equations:

$$x_1 + 2x_2 = 5 \quad (19.27)$$

$$3x_1 + 4x_2 = 6 \quad (19.28)$$

We want to set the equations on the following form:

$$Ax = b \quad (19.29)$$

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:

$$x = A^{-1}b \quad (19.30)$$

We get:

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

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

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

Python Script:

```
1 import numpy as np
2 import numpy.linalg as la
3
4 A = np.array([[1, 2],
5               [3, 4]])
6
7 b = np.array([5],
```

```

8         [6]])
9
10 Ainv = la.inv(A)
11
12 x = Ainv.dot(b)
13
14 print(x)

```

The results becomes:

```

1 [[ -4. ]
2  [ 4.5]]

```

You can also use the following:

```

1 x = np.linalg.solve(A, b)

```

Python Script:

```

1 import numpy as np
2
3 A = np.array([[1, 2],
4               [3, 4]])
5
6 b = np.array([[5],
7               [6]])
8
9 x = np.linalg.solve(A, b)
10 print(x)

```

Note! The A matrix must be square and of full-rank, i.e. the inverse matrix needs to exists.

[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:

```

1 import numpy as np
2
3 A = np.array([[1, 2],
4               [3, 4],
5               [7, 8]])

```

```

6
7 b = np.array ([[5],
8             [6],
9             [9]])
10
11 #x = np.linalg.solve(A, b) #Not working because inverse(A) does not
12 #exists
13
14 x = np.linalg.lstsq(A, b, rcond=None)[0]
15
16 print(x)

```

The results becomes:

```

1 [[-3.5
2 [ 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

Given the following equations:

$$x_1 + 2x_2 = 5 \quad (19.34)$$

$$3x_1 + 4x_2 = 6 \quad (19.35)$$

$$7x_1 + 8x_2 = 9 \quad (19.36)$$

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]

# Chapter 20

## Complex Numbers in Python

### 20.1 Introduction to Complex Numbers

A complex number is defined like this:

$$z = a + jb \quad (20.1)$$

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:

$$z* = a - jb \quad (20.2)$$

You may also represent imaginary numbers on exponential/polar form:

$$z = re^{j\theta} \quad (20.3)$$

Where:

$$r = |z| = \sqrt{a^2 + b^2} \quad (20.4)$$

and

$$\theta = \arctan \frac{b}{a} \quad (20.5)$$

Note that  $a = r \cos \theta$  and  $b = r \sin \theta$

Figure 20.2 shows an illustration of complex numbers on polar form.

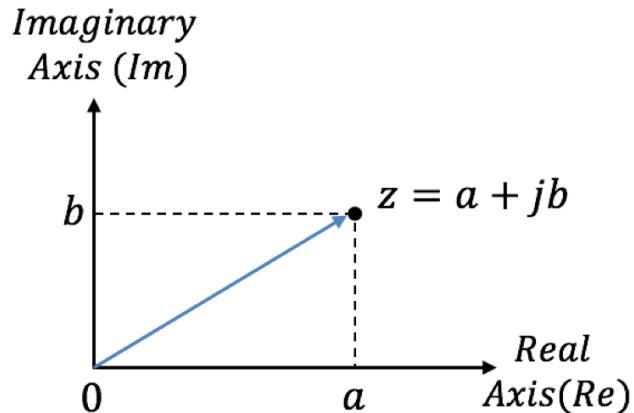


Figure 20.1: Complex Numbers

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:

$$z_1 = r_1 e^{j\theta_1} \quad (20.6)$$

$$z_2 = r_2 e^{j\theta_2} \quad (20.7)$$

Multiplication:

$$z_3 = z_1 z_2 = r_1 r_2 e^{j(\theta_1 + \theta_2)} \quad (20.8)$$

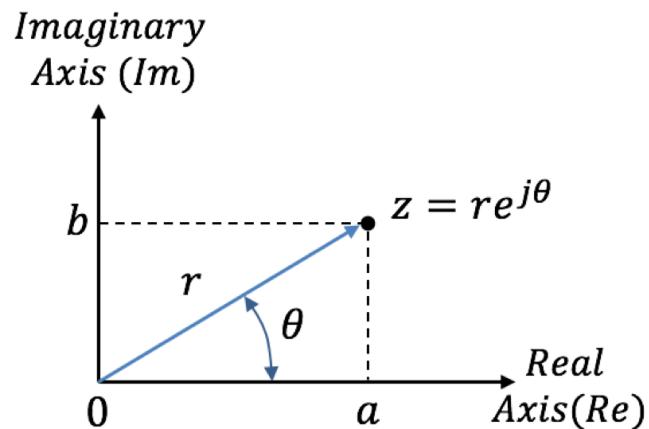


Figure 20.2: Complex Numbers - Polar form

Division:

$$z_3 = \frac{z_1}{z_2} = \frac{r_1 e^{j\theta_1}}{r_2 e^{j\theta_2}} = \frac{r_1}{r_2} e^{j(\theta_1 - \theta_2)} \quad (20.9)$$

## 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:

$$a = 5 + 3j \quad (20.10)$$

$$b = 1 - 1j \quad (20.11)$$

In Python we can define the complex numbers and perform basic operations (+, -, \*, /) like this:

```

1 a = 5 + 3j
2 b = 1 - 1j
3
4 c = a + b
5 print(c)
6
7 d = a - b
8 print(d)
9
10 e = a * b
11 print(e)
12
13 f = a / b
14 print(f)

```

Listing 20.1: Basic Complex Numbers in Python

[End of Example]

### **Example 20.2.2.** Complex Number Functions in Python

```

1 import cmath
2
3 x = 2
4 y = -3
5
6 # converting x and y into complex number using complex()
7 z = complex(x,y)
8 print(z.real)
9 print(z.imag)
10
11 print(z.conjugate())
12
13 # converting complex number into polar using polar()
14 w = cmath.polar(z)
15 print(w)
16
17 # converting complex number into rectangular using rect()
18 w = cmath.rect(2,3)
19 print(w)
20

```

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:

$$\frac{dy}{dt} = f(t, y), y(t_0) = y_0 \quad (21.1)$$

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:

$$\dot{x} = -ax + bu \quad (21.2)$$

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

[End of Example]

With Python 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:

$$\dot{x} = ax \quad (21.3)$$

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:

$$x(t) = e^{at}x_0 \quad (21.4)$$

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:

```
1 import math as mt
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5
6 # Parameters
7 T = 5
8 a = -1/T
9
10 x0 = 1
11 t = 0
12
13 tstart = 0
14 tstop = 25
15
16 increment = 1
17
18 x = []
19 x = np.zeros(tstop+1)
20
21 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.

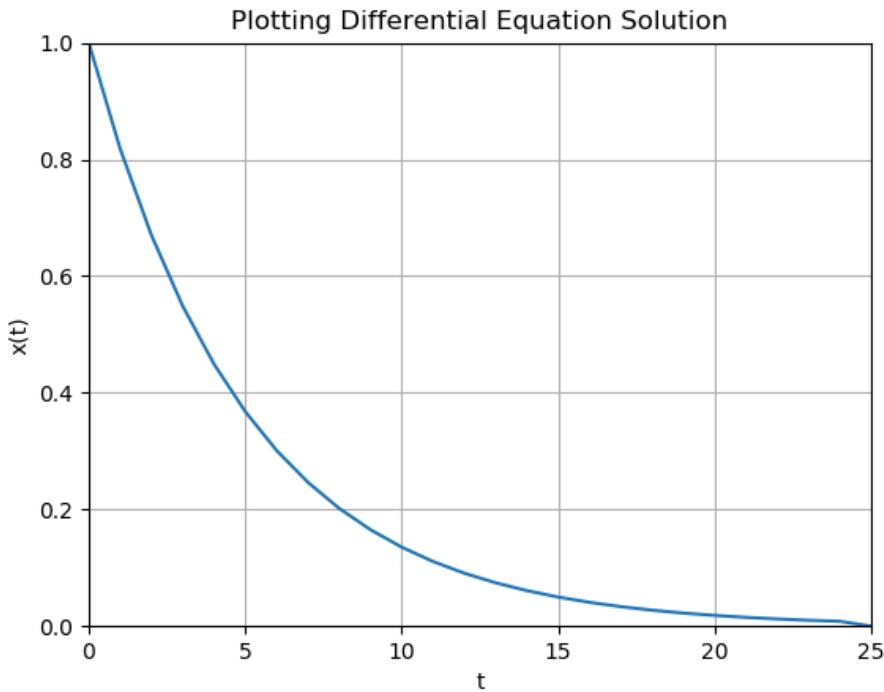


Figure 21.1: Plotting Differential Equation Solution

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

```

```

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 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:

$$\dot{x} = ax \quad (21.5)$$

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:

functionname: Function that returns derivative values at requested x and t values  
as  $dxdt = \text{model}(x,t)$

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
```

```

18     dxdt = a * x
19
20     return dxdt
21
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])

```

```
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 this case  $a$ ) as an input argument. By doing this, it is very easy to change 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:

$$\frac{dx}{dt} = -y \quad (21.6)$$

$$\frac{dy}{dt} = x \quad (21.7)$$

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:

```

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+increment,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?):

$$\frac{dx_1}{dt} = -x_2 \quad (21.8)$$

$$\frac{dx_2}{dt} = x_1 \quad (21.9)$$

The Python code then becomes:

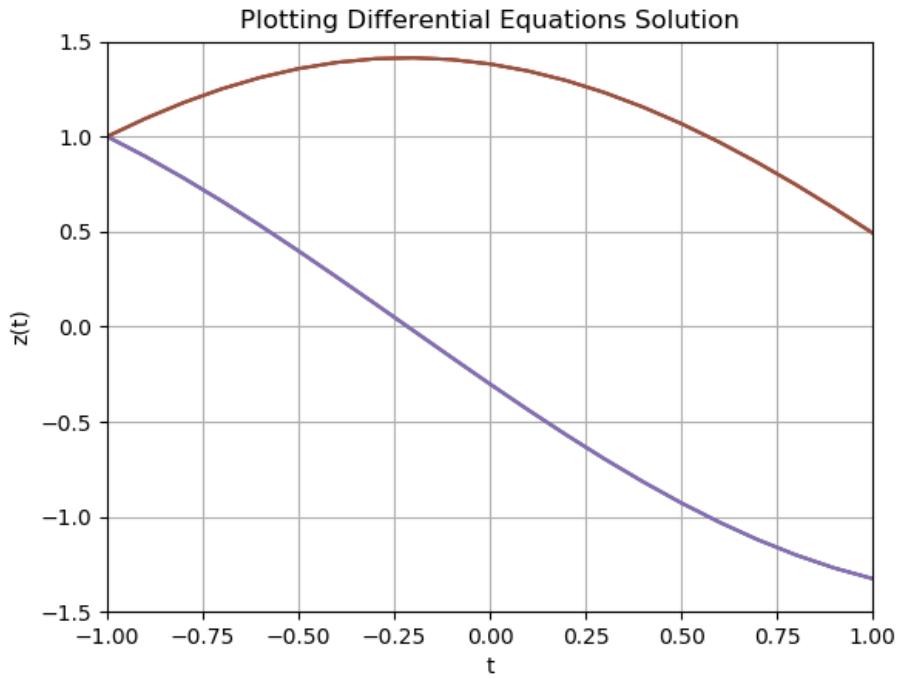


Figure 21.2: Figure Name

```

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+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]

```

```

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()

```

Listing 21.6: xxx

The plot and results will be the same.

[End of Example]

## 21.4 Solving Higher order Differential Equations

We shall use Python to solve and plot the results of the following differential equation:

$$(1 + t^2)\ddot{w} + 2t\dot{w} + 3w = 2 \quad (21.10)$$

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:

$$\ddot{w} = \frac{2 - 2t\dot{w} - 3w}{1 + t^2} \quad (21.11)$$

Note! Higher order differential equations must be reformulated into a system of first order differential equations.

We do the following "trick":

$$w = x_1 \quad (21.12)$$

$$\dot{w} = x_2 \quad (21.13)$$

This gives a set of 1.order differential equations:

$$\dot{x}_1 = x_2 \quad (21.14)$$

$$\dot{x}_2 = \frac{2 - 2tx_2 - 3x_1}{1 + t^2} \quad (21.15)$$

Now we can relatively easy implement the system in Python.

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 = 5
8 increment = 0.1
9
10 # Initial condition
11 x_init = [0,1]
12
13
14 t = np.arange(tstart,tstop+increment)
15
16
17 # Function that returns dx/dt
18 def mydiff(x, t):
19     dx1dt = x[1]
20     dx2dt = (2 - t*x[1] - 3*x[0])/(1 + t**2)
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]
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([0, 5, -1, 2])
42 plt.show()

```

Listing 21.7: xxx

This gives the the plot shown in Figure 21.3.

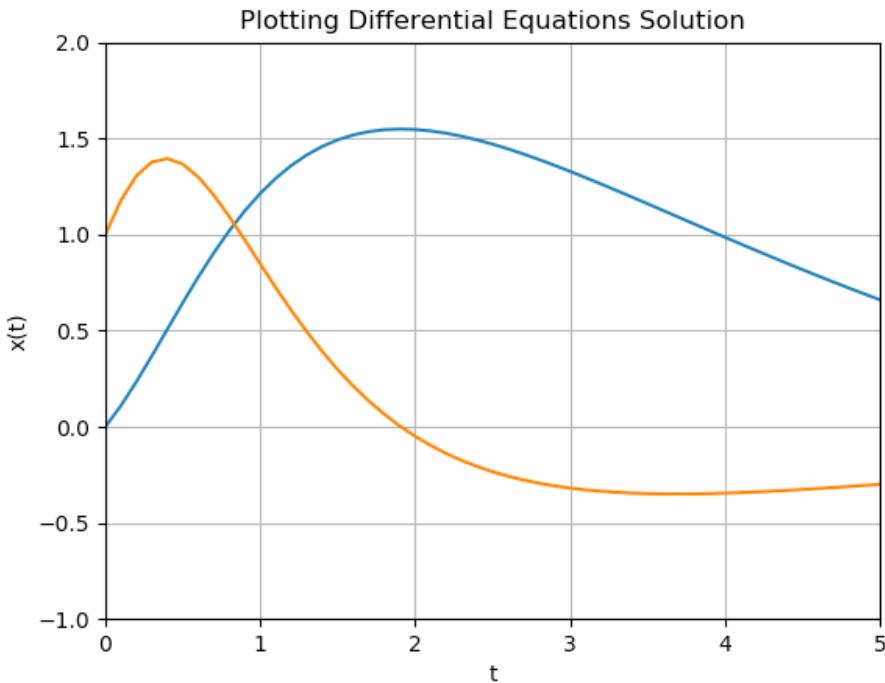


Figure 21.3: 2.order Differential Equation

## 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:

$$\text{birth rate} = bx \quad (21.16)$$

$$\text{death rate} = px^2 \quad (21.17)$$

Then the total rate of change of bacteria population is:

$$\dot{x} = bx - px^2 \quad (21.18)$$

Note!  $\dot{x}$  is the same as  $\frac{dx}{dt}$

Set  $b=1/\text{hour}$  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:

$$\dot{x} = ax + b \quad (21.19)$$

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:

$$\dot{x} = -ax + bu \quad (21.20)$$

Note!  $\dot{x}$  is the same as  $\frac{dx}{dt}$

We have the following:

- $x$  - Process variable, e.g., Level, Pressure, Temperature, etc.

- $u$  - 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.

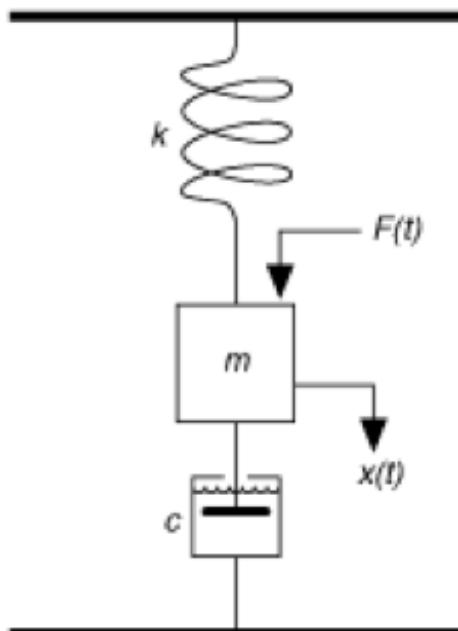


Figure 21.4: Mass-Spring-Damper System

The system can be described by the following equation:

$$F(t) - cx(t) - kx(t) = m\ddot{x}(t) \quad (21.21)$$

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 ( $m$ ).

$\dot{x}(t)$  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  $[t_0, t_f]$ :

$$3w' + \frac{1}{1+t^2}w = \cos(t) \quad (21.22)$$

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:

$$\dot{x}_1 = x_2 \quad (21.23)$$

$$\dot{x}_2 = -\frac{g}{r}x_1 - \frac{b}{mr^2}x_2 \quad (21.24)$$

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.

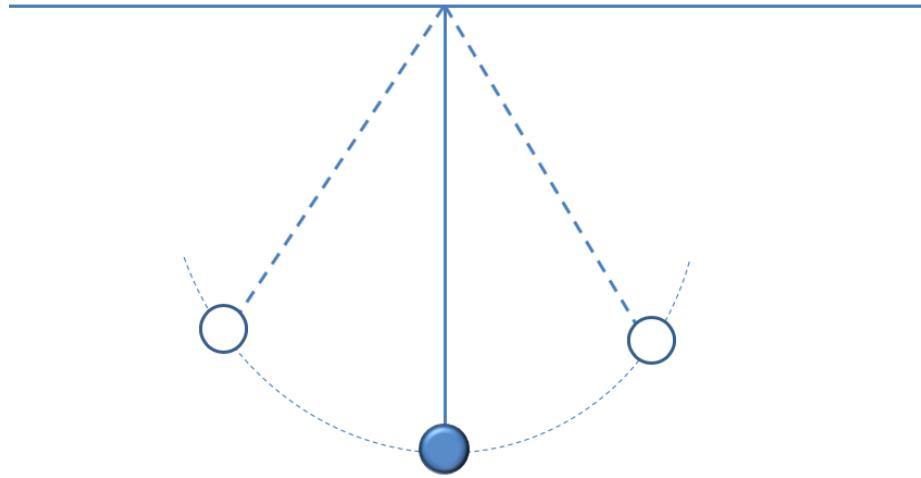


Figure 21.5: Pendulum

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:

$$f(x) = x^2 + 2x + 1 \quad (22.1)$$

We start by plotting the function:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 xmin = -5
5 xmax = 5
6 dx = 0.1
7
8 N = int((xmax - xmin)/dx)
9
10 x = np.linspace(xmin, xmax, N+1)
11 #x = np.arange(xmin, xmax+dx, dx) #Alternative implementation
12
13 y = x**2 + 2*x + 1;
14
15
16 plt.plot(x,y)
17 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:

```
1 from scipy import optimize
2
```

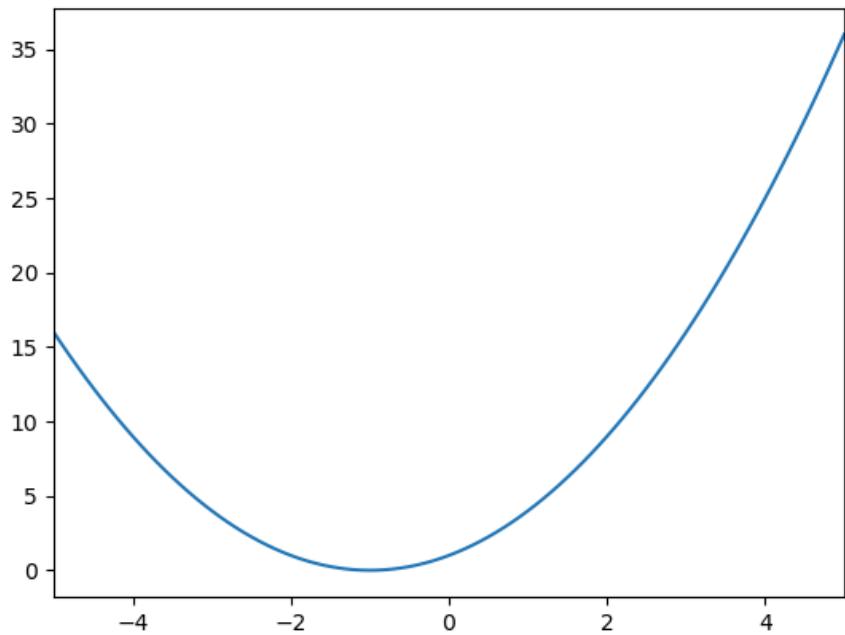


Figure 22.1: Optimization Example

```

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:

$$f(x) = x^2 + 2x + 1 \quad (22.2)$$

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.

[End of Exercise]

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:

$$f(x) = x^2 + 2x + 1 \quad (22.3)$$

Test the different optimization (finding minimum) functions:

`scipy.optimize.fmin`

`scipy.optimize.minimize_scalar`

`scipy.optimize.minimize`

Compare the results.

[End of Exercise]

**Exercise 22.0.3.** Optimization

Given the following function:

$$f(x) = x^3 - 4x \quad (22.4)$$

Test the different optimization (finding minimum) functions:

`scipy.optimize.fminbound`

```
scipy.optimize.fmin  
scipy.optimize.minimize_scalar  
scipy.optimize.minimize
```

Compare the results. You should also plot the function.

[End of Exercise]

**Exercise 22.0.4.** Minimum for function with 2 variables

Given the following function:

$$f(x, y) = 2(x - 1)^2 + x - 2 + (y - 2)^2 + y \quad (22.5)$$

Plot the function and find the minimum.

[End of Exercise]

**Exercise 22.0.5.** Optimization - Rosenbrock's Banana Function

Given the following function:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \quad (22.6)$$

This function is known as Rosenbrock's banana function.

Plot the function and find the minimum.

[End of Exercise]

# **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:

$$\frac{dy}{dt} = f(t, y), y(t_0) = y_0 \quad (24.1)$$

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:

$$\dot{x} = -ax + bu \quad (24.2)$$

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

[End of Example]

With Python 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  $T_s = 0.1s$ .

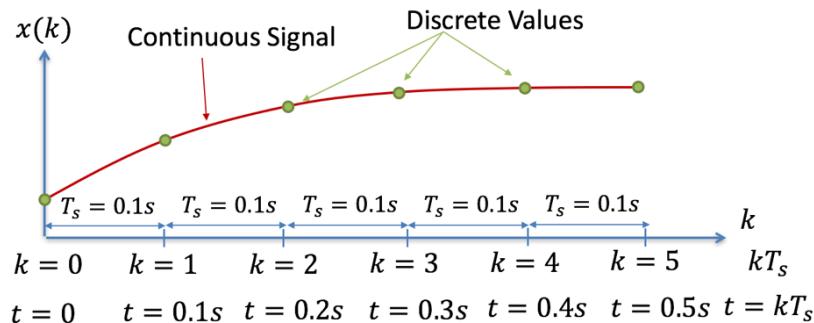


Figure 25.1: Discretization

In order to discretize a continuous model there are lots of different methods to use.

A simple discretization method is the Euler Forward method:

$$\dot{x} = \frac{x(k+1) - x(k)}{T_s} \quad (25.1)$$

$T_s$  is the Sampling Time

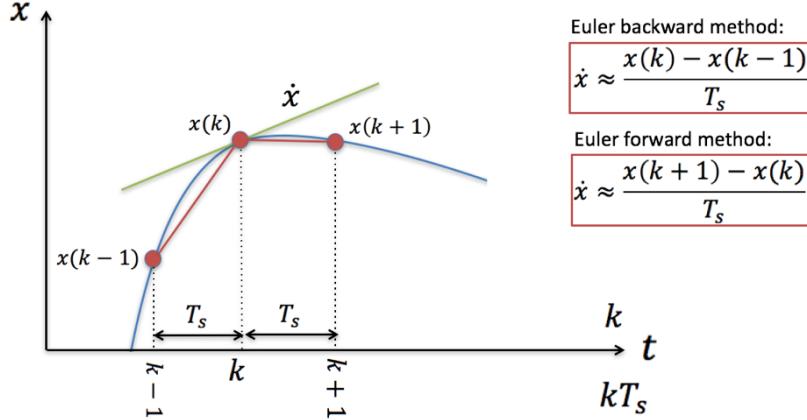


Figure 25.2: Euler Discretization methods

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:

$$\dot{x} = -ax + bu \quad (25.2)$$

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:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s} \quad (25.3)$$

$T_s$  - Sampling Interval

Then we get:

$$\frac{x_{k+1} - x_k}{T_s} = -ax_k + bu_k \quad (25.4)$$

This gives the following discrete differential equation:

$$x_{k+1} = (1 - T_s a)x_k + T_s b u_k \quad (25.5)$$

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:

```

1 # Simulation of discrete model
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 N = int(Tstop/Ts) # Simulation length
15 data = []
16 data.append(xk)
17
18
19 # Simulation
20 for k in range(N):
21     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 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.

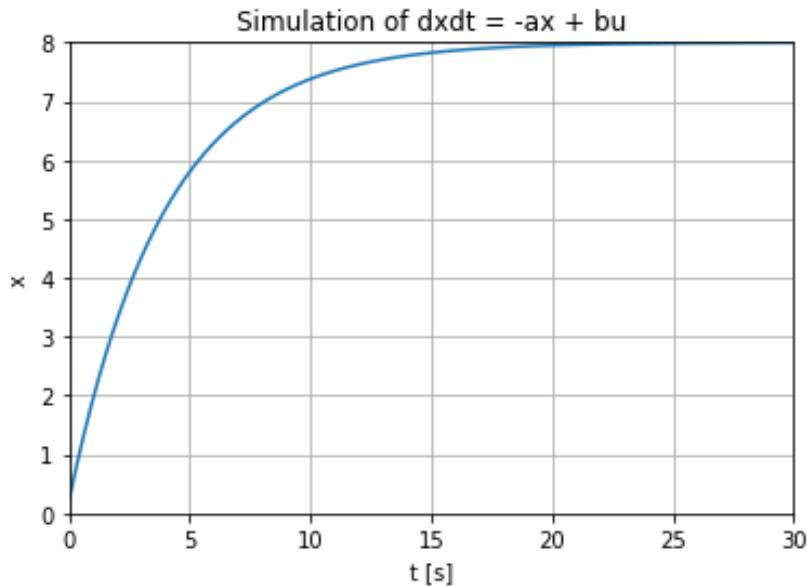


Figure 25.3: Simulation of Discrete System using Python

Here is an alternative solution presented:

```

1 # Simulation of discrete model
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 # Sampling Time
11 Tstop = 30 # End of Simulation Time
12 uk = 1 # Step Response
13 N = int(Tstop/Ts) # Simulation length
14 x = np.zeros(N+2) # Initialization the x vector
15 x[0] = 0
16
17 # Simulation
18 for k in range(N+1):
19     x[k+1] = (1 - a*Ts) * x[k] + Ts * b * uk
20
21
22 # Plot the Simulation Results
23 t = np.arange(0, Tstop+2*Ts, Ts) #Create the Time Series
24 plt.plot(t,x)
25
26 # Formatting the appearance of the Plot
27 plt.title('Simulation of dxdt = -ax + bu')
28 plt.xlabel('t [s]')
29 plt.ylabel('x')
```

Name	Type	Size	Value
N	int	1	300
Ts	float	1	0.1
Tstop	int	1	30
a	float	1	0.25
b	int	1	2
data	list	301	[0, 0.2, 0.395, 0.585125, 0.7704968750000001, 0.9512344531250001, 1.12...
k	int	1	299
t	float64	(301,)	[ 0. 0.1 0.2 ... 29.8 29.9 30.]
uk	int	1	1
xk	float	1	7.995977697799602
xk1	float	1	7.995977697799602

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:

$$\text{birth rate} = bx \quad (25.6)$$

$$\text{death rate} = px^2 \quad (25.7)$$

Then the total rate of change of bacteria population is:

$$\dot{x} = bx - px^2 \quad (25.8)$$

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:

$$\frac{dx_1}{dt} = -x_2 \quad (25.9)$$

$$\frac{dx_2}{dt} = x_1 \quad (25.10)$$

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:

$$x_{k+1} = (1 - T_s a)x_k + T_s b u_k \quad (26.1)$$

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:

```
1 # Simulation of discrete model
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 N = int(Tstop/Ts) # Simulation length
15 data = []
16 data.append(xk)
17
18
19 # Simulation
20 for k in range(N):
21     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.

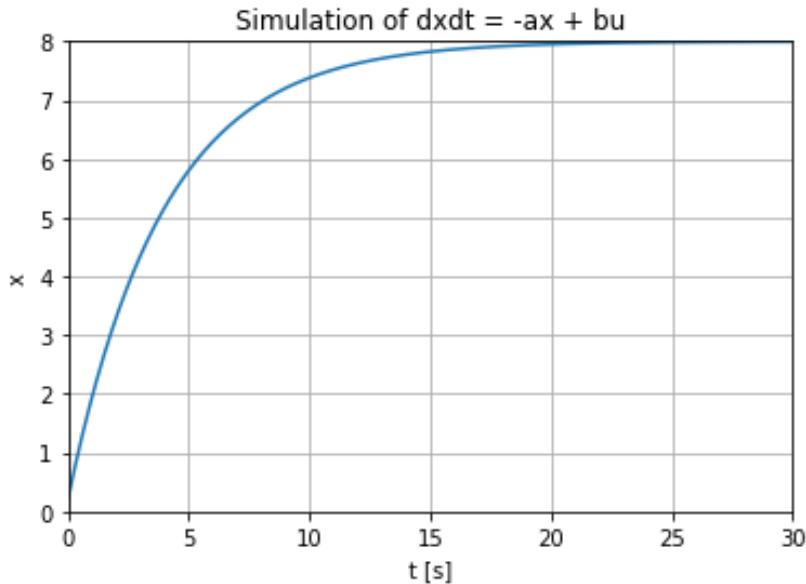


Figure 26.1: Simulation of Discrete System using Python

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:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 plt.axis([0, 10, 0, 1])
5
6 delay = 1 #Seconds
7
8 for i in range(10):
9     y = np.random.random()
10    plt.scatter(i, y)
11    plt.pause(delay)
12
13 plt.show()
```

Listing 26.2: Real-Time Plotting in Python

We get the following plot as shown in Figure 27.1.

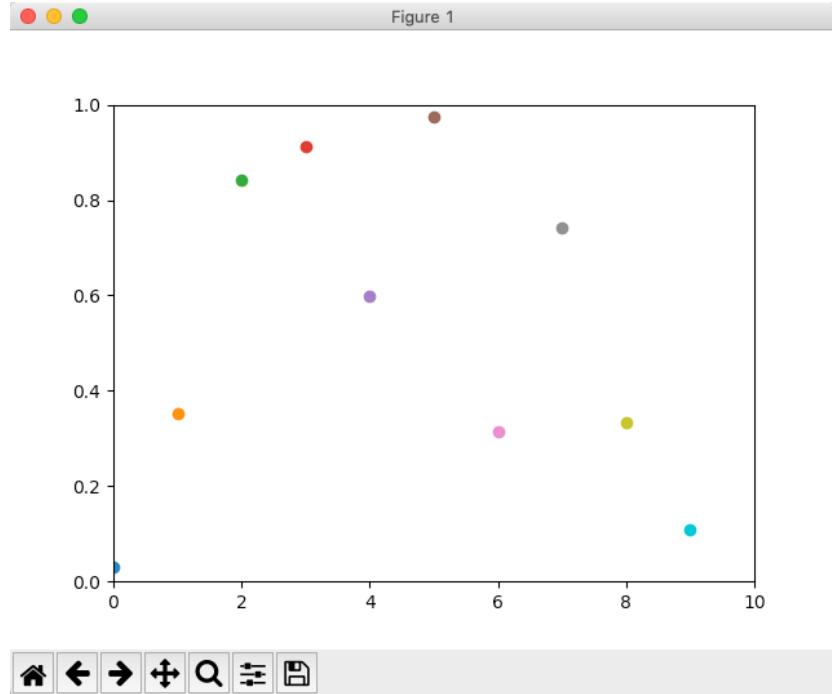


Figure 26.2: Real-Time Plotting with Python

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 Spyder, you typically need to change the the settings for how graphics 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.

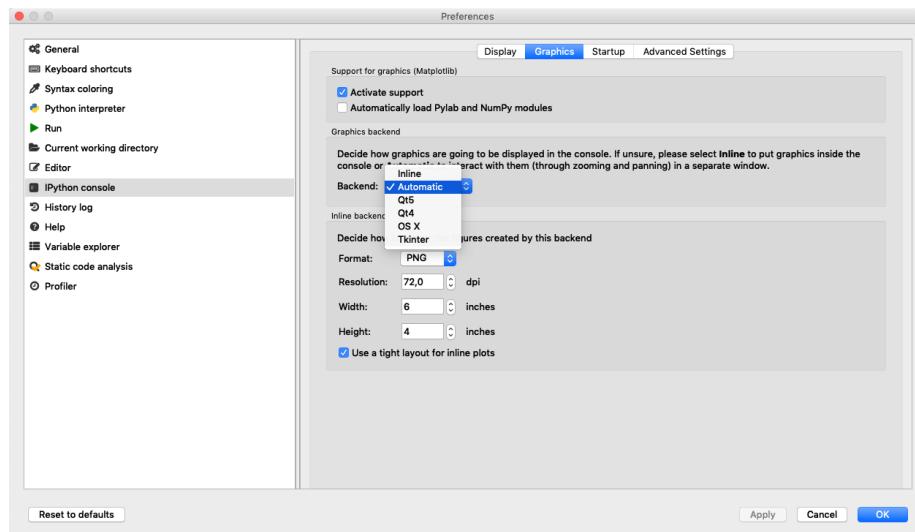


Figure 26.3: Change how Graphics are displayed in the Spyder Editor

### Example 26.2.2. Real-Time Simulation of Discrete System

Lets change the discrete simulation example presented in the beginning of this chapter:

```
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 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.

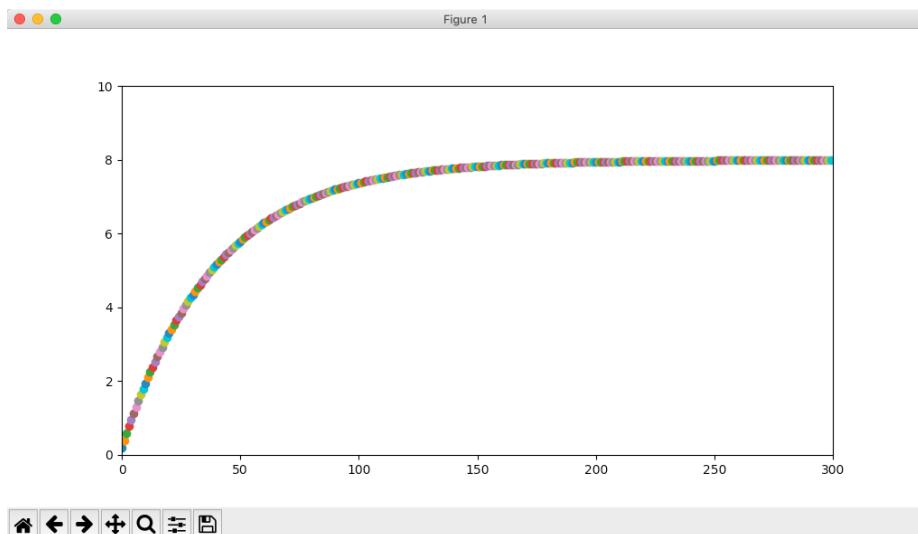


Figure 26.4: Real-Time Simulation of Discrete System

You cannot see the 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 that automatically scales the x axis and the y axis. What's best depends on if you know the simulation length from the beginning and know the minimum and maximum value of the simulation results.

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.

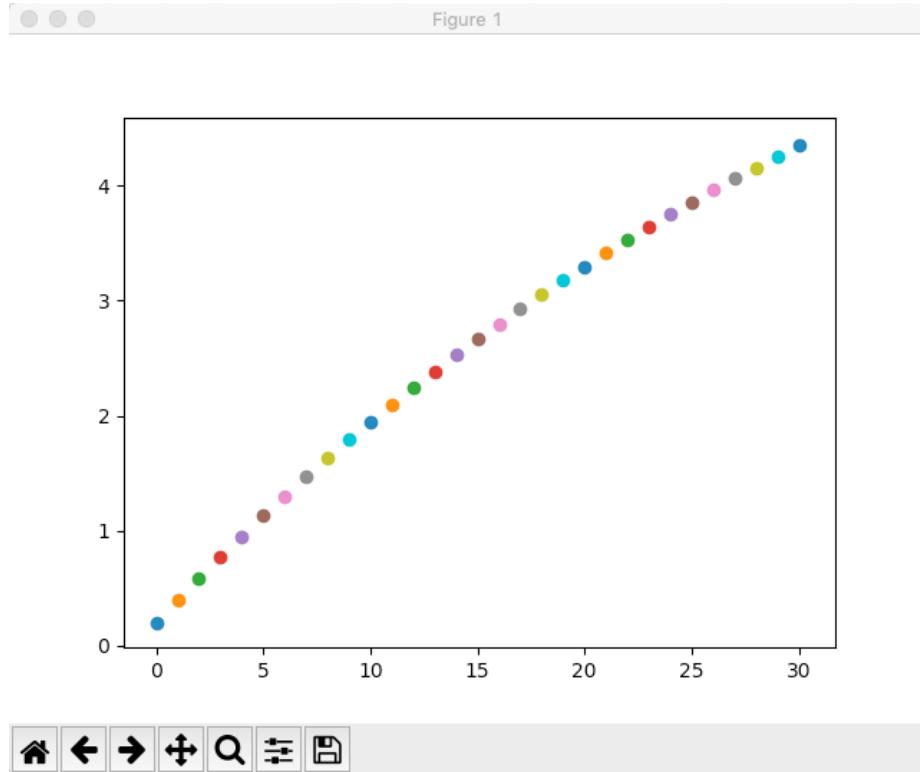


Figure 26.5: Real-Time Simulation of Discrete System with Auto-scaling

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)

```

```

18 plt.axis([0, N*Ts, 0, y_scale_max])
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     t = Ts*k
26
27     plt.scatter(t, xk1)
28     plt.pause(Ts)
29
30 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.

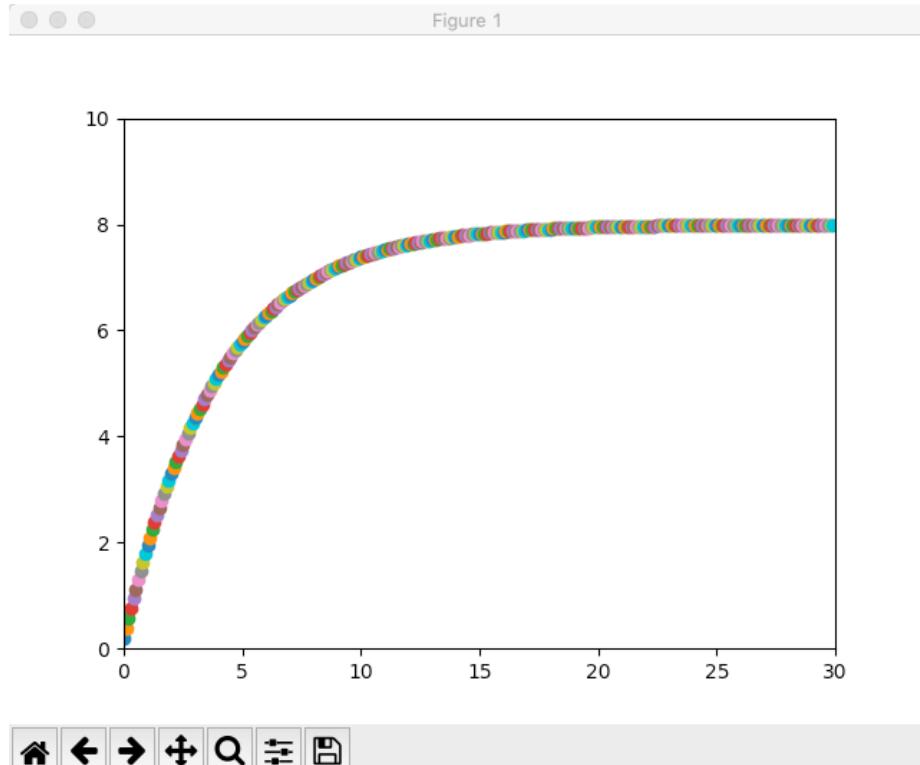


Figure 26.6: Real-Time Simulation of Discrete System

We can also add some formatting regarding the appearance of the plot (xlabel, ylabel, title, etc.). The final code example is shown below.

```

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.

```
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     temp_c = round(np.random.random() , 2)  
16  
17     # Add x and y to lists  
18     xs.append(dt.datetime.now().strftime('%H:%M:%S.%f'))  
19     ys.append(temp_c)  
20  
21     # Limit x and y lists to 20 items  
22     xs = xs[-20:]  
23     ys = ys[-20:]  
24  
25     # Draw x and y lists  
26     ax.clear()  
27     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),
38                               interval=1000)
39 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.

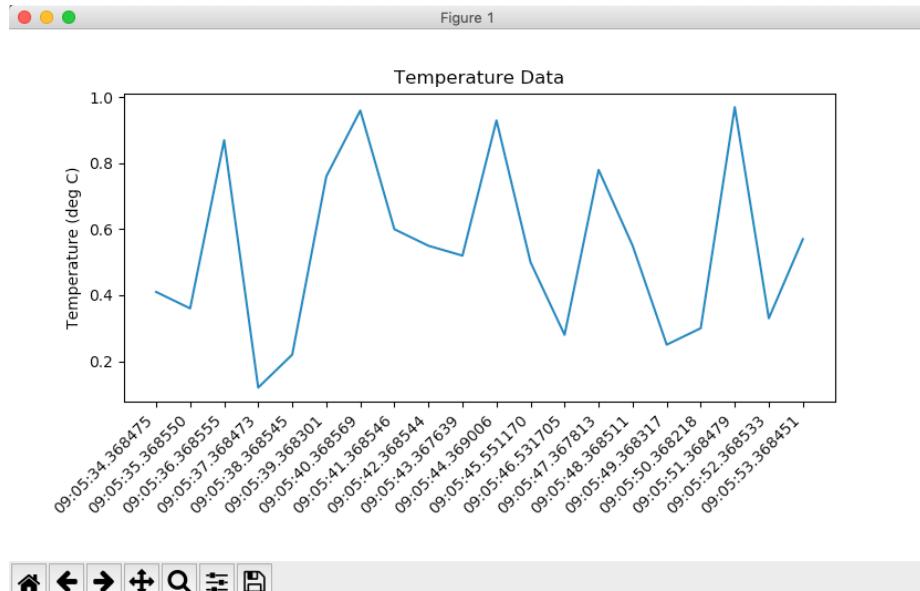


Figure 26.7: Real-Time Plotting with Animation

[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
24     # a gobal variable
25
26     # Model of Discrete System
27     xk1 = (1 - a*Ts) * xk + Ts * b * uk
28     xk = xk1
29
30     # Add x and y to lists
31     xs.append(dt.datetime.now().strftime('%H:%M:%S.%f'))
32     ys.append(xk1)
33
34     N = 60 # Limit x and y lists to N items
35     xs = xs[-N:]
36     ys = ys[-N:]
37
38     # Draw x and y lists
39     ax.clear()
40     ax.plot(xs, ys)
41
42     # Format plot
43     plt.xticks(rotation=45, ha='right')
44     plt.subplots_adjust(bottom=0.30)
45     plt.title('Simulation of dxdt = -ax + bu')
46     plt.xlabel('t [s]')
47     plt.ylabel('x')
48     plt.grid()
49
50 # Set up plot to call animate() function periodically
51 ani = animation.FuncAnimation(fig, simulation, fargs=(xs, ys),
52                             interval=100)
53 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.

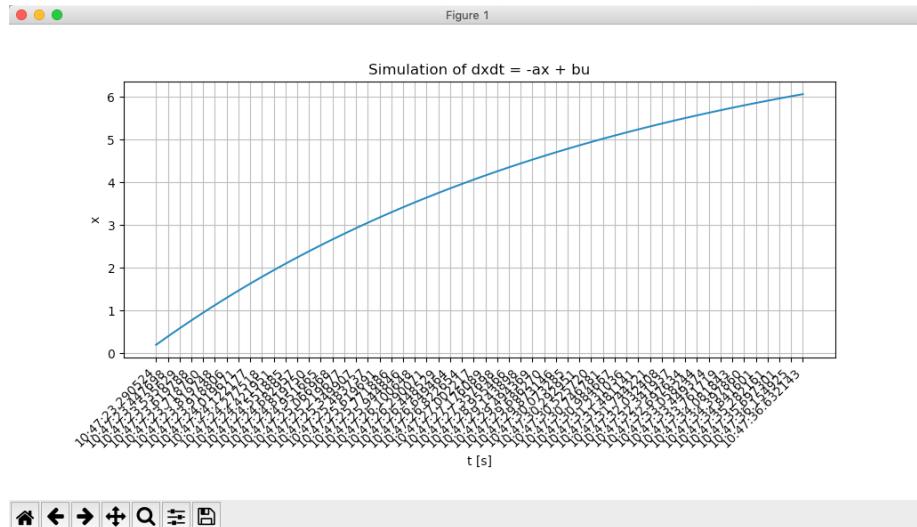


Figure 26.8: Real-Time Simulation of Discrete Dynamic System with Animation

[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 what's most important for you in your simulations.

**Example 26.3.3.** Real-Time Plotting with Animation with improved Performance

Python Code:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib.animation as animation
4
5 # Parameters
6 x_len = 200      # Number of points to display

```

```

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
29     between 0 and 20
30
31     temp_c = round(rand_val, 2)
32
33     #print (temp_c)
34
35     # Add y to list
36     ys.append(temp_c)
37
38     # Limit y list to set number of items
39     ys = ys[-x_len:]
40
41     # Update line with new Y values
42     line.set_ydata(ys)
43
44     return line,
45
46 # Set up plot to call animate() function periodically
47 ani = animation.FuncAnimation(fig,
48     animate,
49     fargs=(ys,),
50     interval=100,
51     blit=True)
52 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

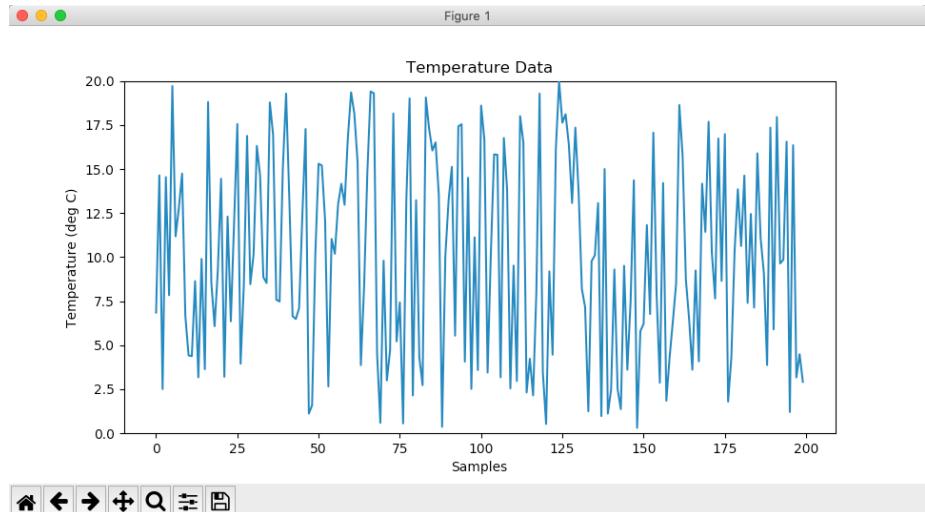


Figure 26.9: Real-Time Plotting with Animation

Python Code:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib.animation as animation
4
5 # Parameters
6 N = 200
7 x_len = N          # Number of points to display
8 y_range = [0, 10]  # Range of possible Y values to display
9
10 # Create figure for plotting
11 fig = plt.figure()
12 ax = fig.add_subplot(1, 1, 1)
13 xs = list(range(0, N))
14 ys = [0] * x_len
15 ax.set_ylim(y_range)
16
17 xk = 0 # Initial Value for x
18
19 # Create a blank line. We will update the line in animate
20 line, = ax.plot(xs, ys)
21
22 # Add labels
23 plt.title('Simulation of dxdt = -ax + bu')
24 plt.xlabel('Samples')
25 plt.ylabel('x')
26
27 # This function is called periodically from FuncAnimation
28 def simulation(i, ys):
29
30     # Model Parameters
31     a = 0.25
32     b = 2
33
34     # Simulation Parameters
35     Ts = 0.1 # Sampling Time
36     uk = 1 # Step Response

```

```

37     global xk # Since we change x inside a function we define it as
38     a gobal variable
39
40     # Model of Discrete System
41     xk1 = (1 - a*Ts) * xk + Ts * b * uk
42     xk = xk1
43
44     # Add y to list
45     ys.append(xk1)
46
47     # Limit y list to set number of items
48     ys = ys[-x_len:]
49
50     # Update line with new Y values
51     line.set_ydata(ys)
52
53     return line,
54
55 # Set up plot to call animate() function periodically
56 ani = animation.FuncAnimation(fig,
57     simulation,
58     fargs=(ys,),
59     interval=100,
60     blit=True)
61 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\\_Animations.html](https://scipy-cookbook.readthedocs.io/items/Matplotlib_Animations.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>

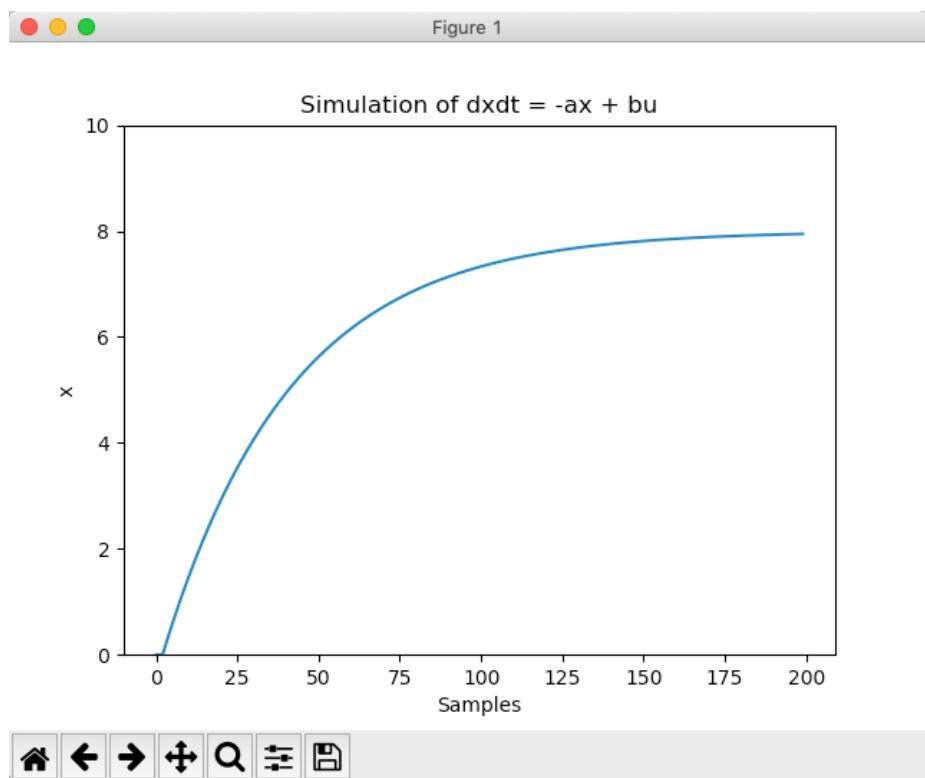


Figure 26.10: Real-Time Plotting with Animation

# **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 your 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:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 plt.axis([0, 10, 0, 1])
5
6 delay = 1 #Seconds
7
8 for i in range(10):
9     y = np.random.random()
10    plt.scatter(i, y)
11    plt.pause(delay)
```

```
12 plt.show()  
13
```

Listing 27.1: Real-Time Plotting in Python

We get the following plot as shown in Figure 27.1.

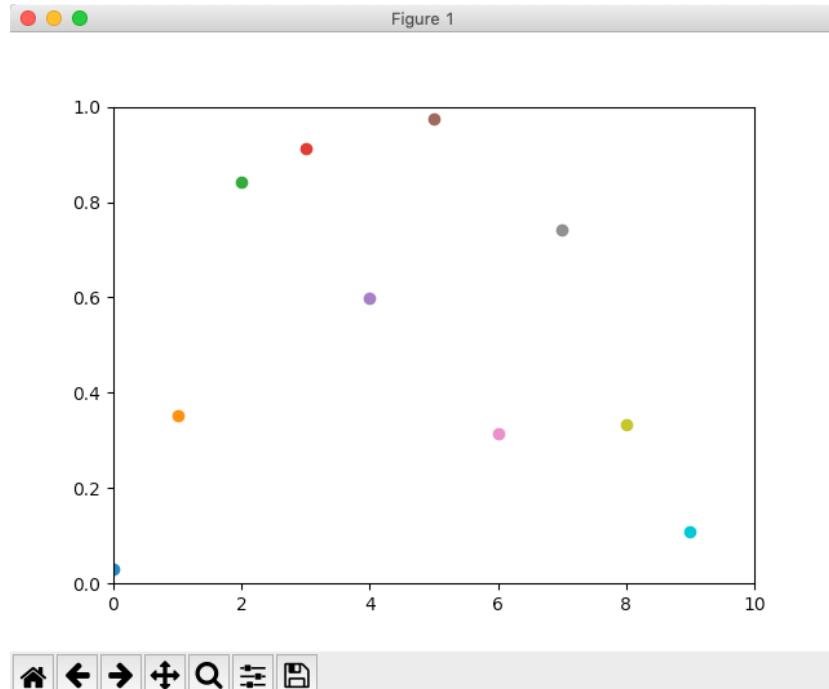


Figure 27.1: Real-Time Plotting with Python

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.

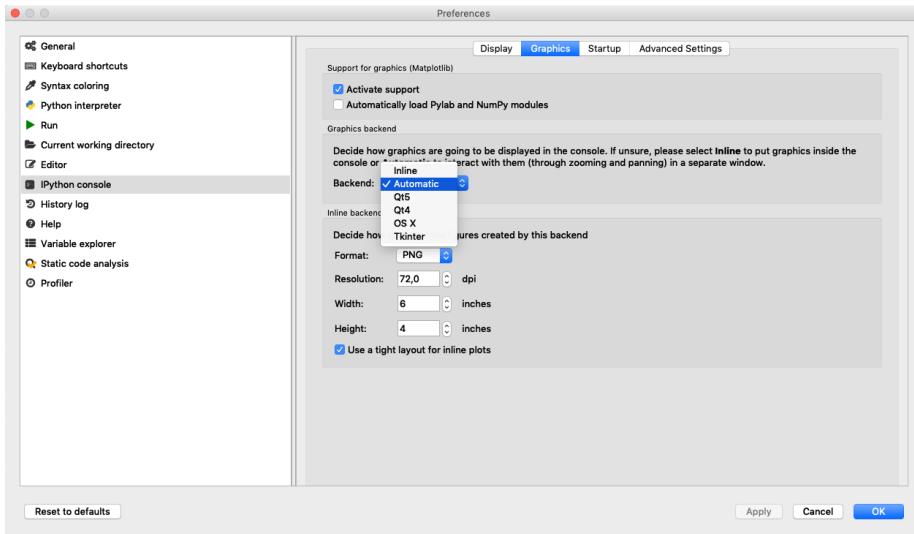


Figure 27.2: Change how Graphics are displayed in the Spyder Editor

### 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:

```
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 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.

```
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),
38                             interval=1000)
39 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]

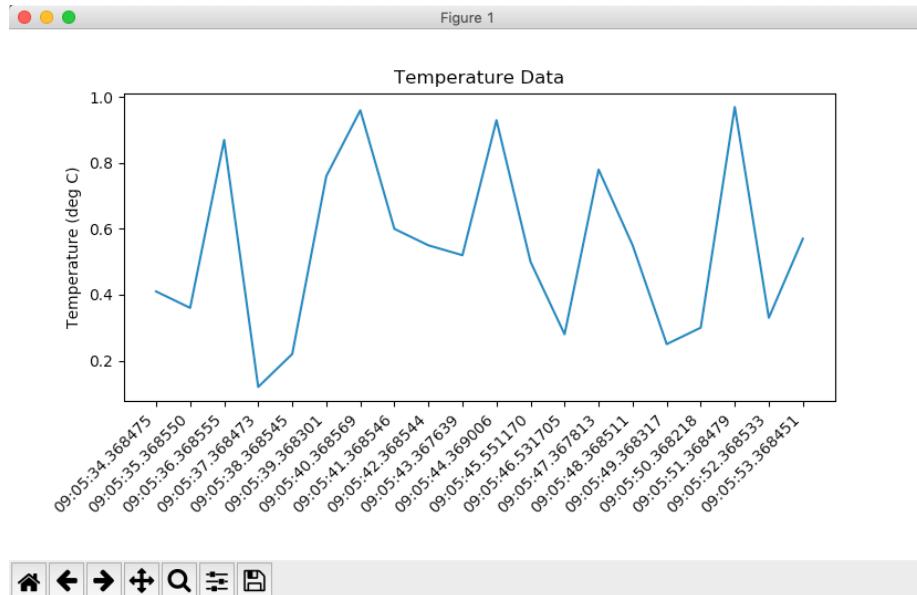


Figure 27.3: Real-Time Plotting with Animation

### 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 what's most important for your own simulations.

**Example 27.3.2.** Real-Time Plotting with Animation with improved Performance

Python Code:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib.animation as animation
4
5 # Parameters
6 x_len = 200          # Number of points to display
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
29     between 0 and 20
30
31     temp_c = round(rand_val, 2)
32
33     #print (temp_c)
34
35     # Add y to list
36     ys.append(temp_c)
37
38     # Limit y list to set number of items
39     ys = ys[-x_len:]
40
41     # Update line with new Y values
42     line.set_ydata(ys)
43
44     return line,
45
46 # Set up plot to call animate() function periodically
47 ani = animation.FuncAnimation(fig,
48     animate,
49     fargs=(ys,),
50     interval=100,
51     blit=True)
52 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\\_Animations.html](https://scipy-cookbook.readthedocs.io/items/Matplotlib_Animations.html)

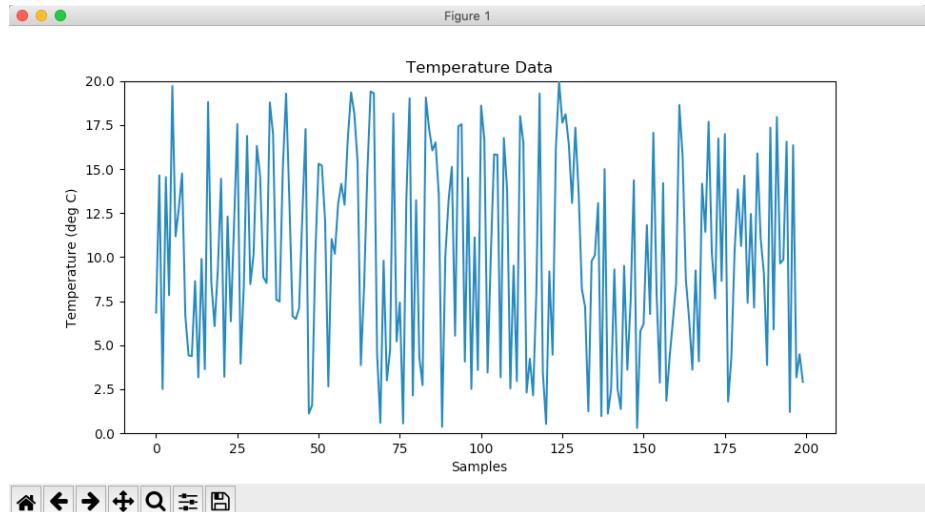


Figure 27.4: Real-Time Plotting with Animation

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 your 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.

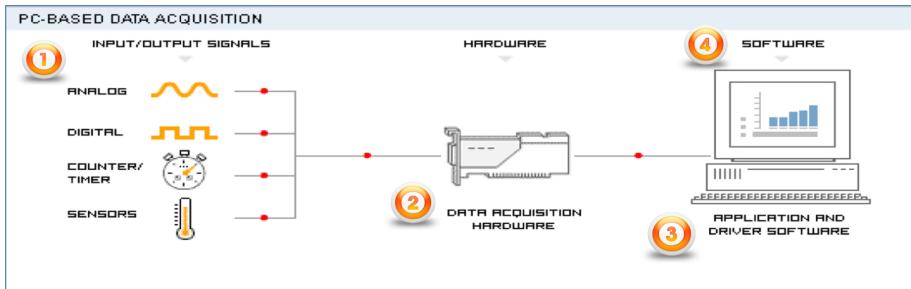


Figure 28.1: Data Acquisition (DAQ) System

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.

A Python API for interacting with NI-DAQmx (GitHub):  
<https://github.com/ni/nidaqmx-python>



Figure 28.2: USB-6008

For more information about NI DAQ:  
[ni.com/daq](http://ni.com/daq)

For more information about Python Resources for NI Hardware and Software:  
[ni.com/python](http://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 NI DAQmx 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).

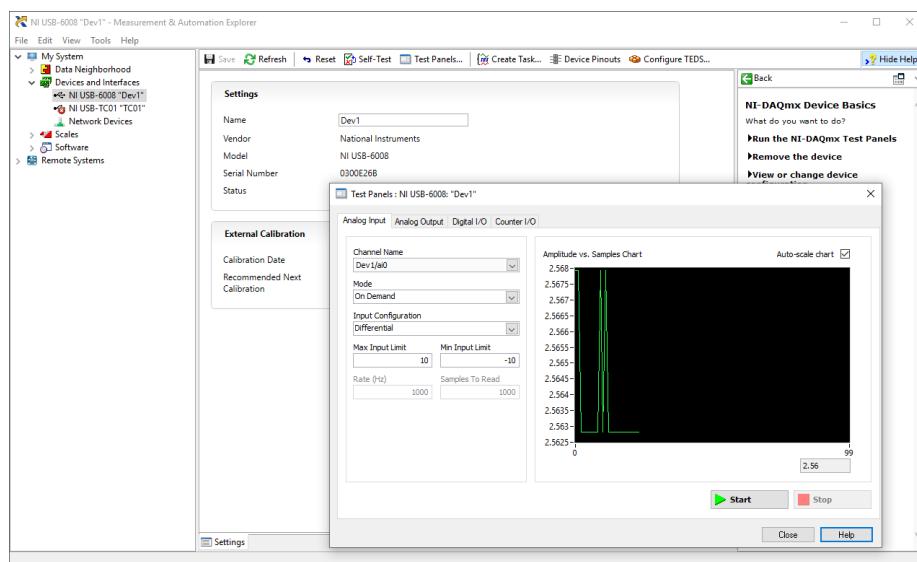


Figure 28.3: Measurement Automation Explorer(MAX)

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:

```
1 import nidaqmx
2
3 with nidaqmx.Task() as task:
4     task.ao_channels.add_ao_voltage_chan('Dev1/ao0', 'mychannel',
5                                         0, 5)
6
7     value = 3
8     task.start()
9     task.write(value)
10    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]

### 28.3.2 Analog Read

**Example 28.3.2.** Analog Read using NI DAQ Device

Python code:

```

1 import nidaqmx
2
3 with nidaqmx.Task() as task:
4     task.ai_channels.add_ai_voltage_chan("Dev1/ai1")
5
6     value = task.read()
7     print(value)
8     task.stop

```

Listing 28.2: Analog Read using NI DAQ Device

[End of Example]

#### **Example 28.3.3.** Analog Read with RSE

Python code:

```

1 import nidaqmx
2
3 from nidaqmx.constants import (
4     TerminalConfiguration)
5
6 with nidaqmx.Task() as task:
7     task.ai_channels.add_ai_voltage_chan("Dev1/ai0",
8         terminal_config=TerminalConfiguration.RSE)
9
10    value = task.read()
11    print(value)
12    task.stop

```

Listing 28.3: Analog Read with RSE

[End of Example]

#### **Example 28.3.4.** Analog Read with Differential

Python code:

```

1 import nidaqmx
2
3 from nidaqmx.constants import (
4     TerminalConfiguration)
5
6 with nidaqmx.Task() as task:
7     task.ai_channels.add_ai_voltage_chan("Dev1/ai0",
8         terminal_config=TerminalConfiguration.DIFFERENTIAL)
9
10    value = task.read()
11    print(value)
12    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:

```
1 import nidaqmx
2
3 with nidaqmx.Task() as task:
4     task.do_channels.add_do_chan("Dev1/port0/line0")
5
6     value = True
7     task.start()
8     task.write(value)
9     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:

```
1 import nidaqmx
2
3 with nidaqmx.Task() as task:
4     task.di_channels.add_di_chan("Dev1/port0/line0")
5
6     task.start()
7     value = task.read()
8     print(value)
9     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_readers.html#module-nidaqmx.stream_readers)

[https://nidaqmx-python.readthedocs.io/en/latest/stream\\_writers.html#module-nidaqmx.stream\\_writers](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.

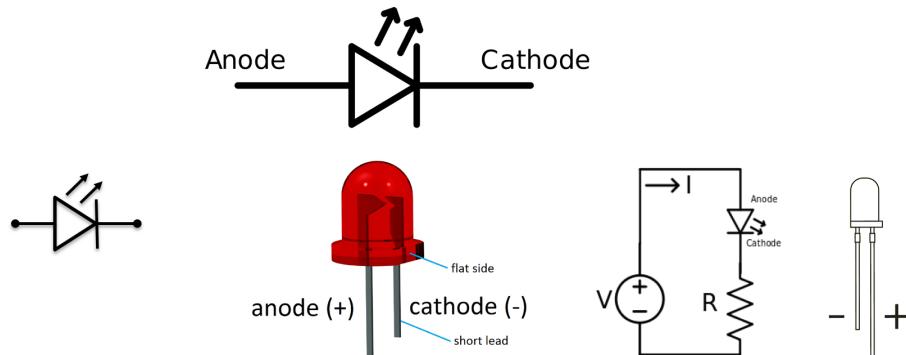


Figure 28.4: LED Overview

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:

```
1 import nidaqmx
2
3 with nidaqmx.Task() as task:
4     task.do_channels.add_do_chan("Dev1/port0/line0")
5
6     value = True
7     task.start()
8     task.write(value)
9     task.stop()
```

Listing 28.7: Turn on a LED using Python

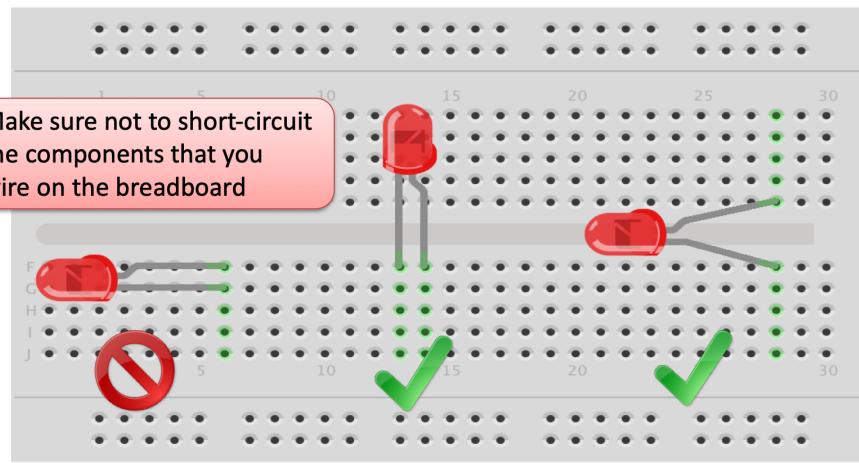


Figure 28.5: How to wire a LED on a Breadboard

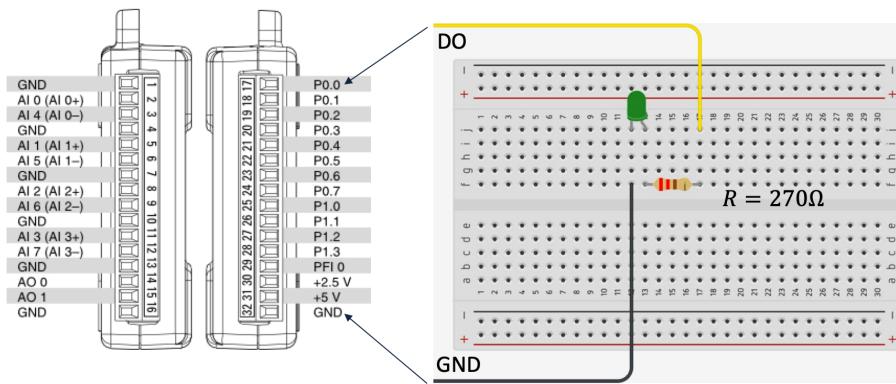


Figure 28.6: Wire the LED and connect to the DAQ device

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:

```

1 import nidaqmx
2 import time
3
4
5 with nidaqmx.Task() as task:
6     task.do_channels.add_do_chan("Dev1/port0/line0")
7
8     value = True
9     task.start
10
11    i = 1
12    while i < 10:
13

```

```

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 Temperature 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:

$$(x_1, y_1) = (0.75V, 25^\circ) \quad (28.1)$$

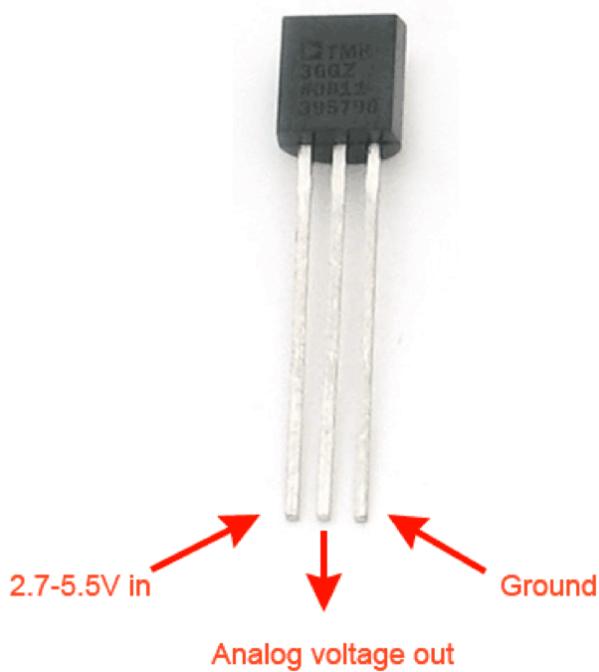


Figure 28.7: TMP36 Temperature Sensor

$$(x_2, y_2) = (1V, 50^\circ) \quad (28.2)$$

From the Datasheet (Figure 28.9) we see that there is a linear relationship between Voltage and degrees Celsius (28.3):

$$y = ax + b \quad (28.3)$$

We can find a and b using the following known formula (28.4):

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1} (x - x_1) \quad (28.4)$$

By putting (28.1) and (28.2) into (28.4) we get:

$$y - 25 = \frac{50 - 25}{1 - 0.75} (x - 0.75) \quad (28.5)$$

Then we get the following formula we can implement in our Python program:

$$y = 100x - 50 \quad (28.6)$$

#### **Example 28.5.1.** Read TMP36 Temperature Data

Python code:

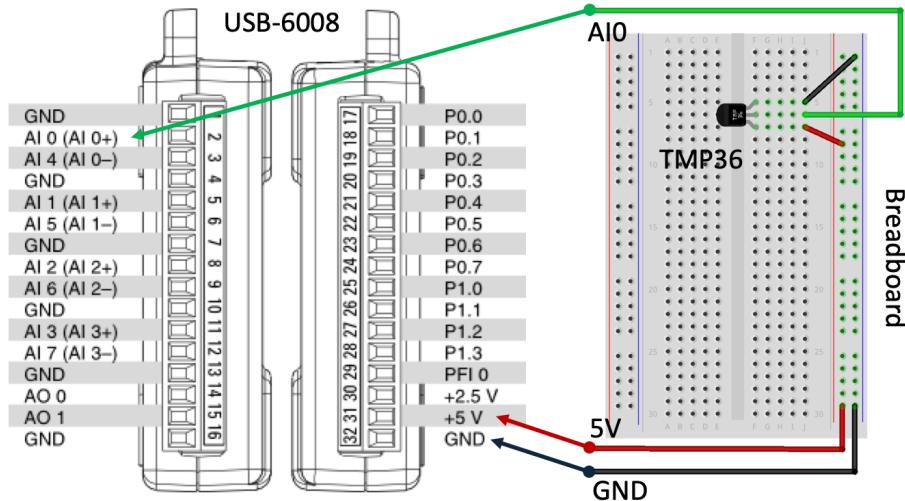


Figure 28.8: TMP36 tmp36 Wiring

```

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",
10         terminal_config=TerminalConfiguration.RSE)
11
12     i = 0
13     while i < 10:
14
15         voltage = task.read()
16
17         degreesC = 100*voltage - 50
18
19         print("Sample:", i)
20         print("Voltage Value:", round(voltage, 2))
21         print("Celsius Value:", round(degreesC, 1))
22         print("\n")
23         time.sleep(1)
24         i = i+1
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

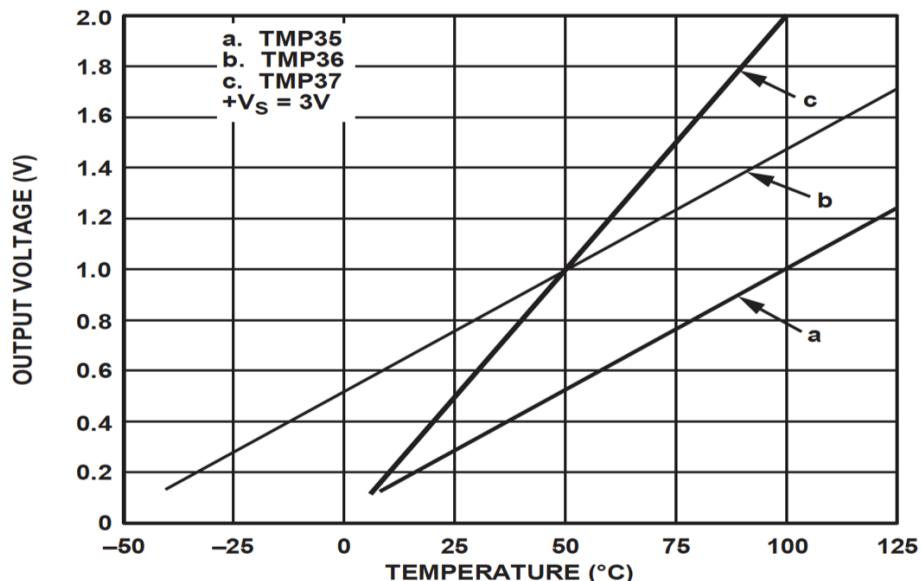


Figure 28.9: TMP3x Datasheet

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:

```

1 import nidaqmx
2 import time
3 import datetime as dt
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import matplotlib.animation as animation
7
8 from nidaqmx.constants import (
9     TerminalConfiguration)
10
11
12 # Create figure for plotting
13 fig = plt.figure()
14 ax = fig.add_subplot(1, 1, 1)
15 xs = []
16 ys = []
17
18 # Initialize DAQ device
19 task = nidaqmx.Task()
20 task.ai_channels.add_ai_voltage_chan("Dev1/ai0", terminal_config=
    TerminalConfiguration.RSE)

```

```

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),
56                               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

- 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 a non-linear relationship between resistance and excitement. To find the temperature we can use the following equation (Steinhart-Hart equation):

$$\frac{1}{T_K} = A + B \ln(R) + C(\ln(R))^3 \quad (28.7)$$

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:

$$T_K = \frac{1}{A + B \ln(R) + C(\ln(R))^3} \quad (28.8)$$

The Temperature in degrees Celsius will then be:

$$T_C = T_K - 273.15 \quad (28.9)$$

Wiring...

Figure 28.10 shows how we wire the components together.

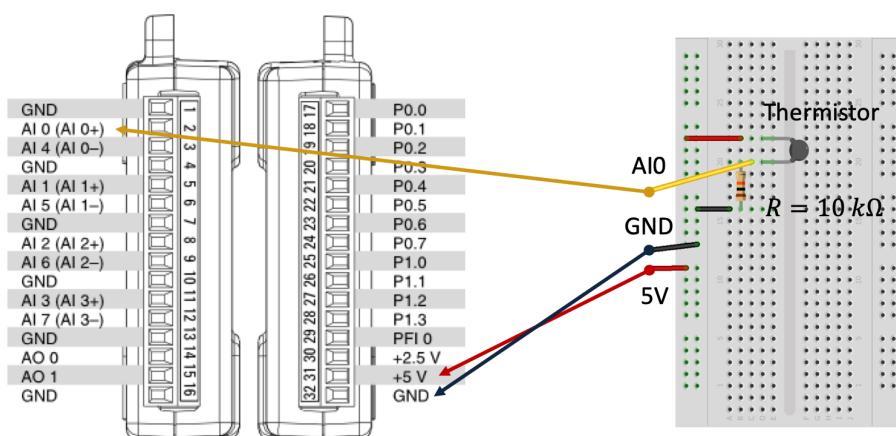


Figure 28.10: Thermistor Wiring

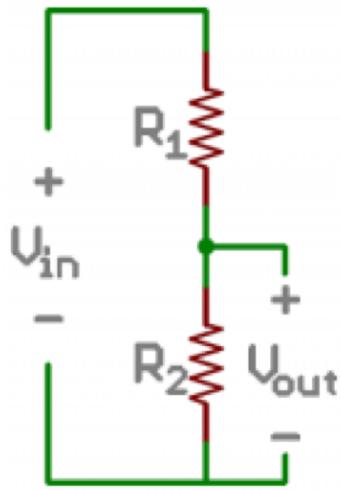


Figure 28.11: Voltage Divider

The wiring is called a "Voltage divider".

Figure 28.11 shows a general Voltage Divider.

A general Voltage Divider has the following equation:

$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2} \quad (28.10)$$

The Voltage Divider for our system becomes as shown in Figure 28.12.

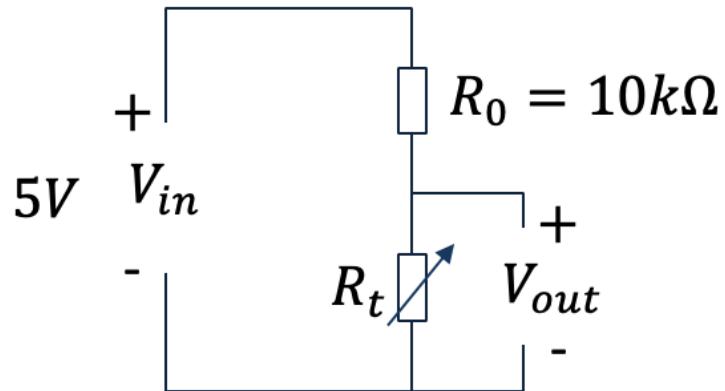


Figure 28.12: Voltage Divider for our System

We then have the following equation:

$$V_{out} = V_{in} \frac{R_t}{R_0 + R_t} \quad (28.11)$$

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:

$$R_t = \frac{V_{out} R_0}{V_{in} - V_{out}} \quad (28.12)$$

We have now 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",
15                                         terminal_config=TerminalConfiguration.RSE)
16
17     i = 0
18     while i < 10:
19
20         Vout = task.read()
21
22         Rt = (Vout*Ro)/(Vin-Vout) # Voltage Divider Equation
23         # Rt=10000; Used for Testing. Setting Rt=10k should give
TempC=25

```

```

24     # Steinhart constants
25     A = 0.001129148
26     B = 0.000234125
27     C = 0.000000876741
28
29     # Steinhart-Hart Equation
30     TempK = 1 / (A + (B * np.log(Rt)) + (C * pow(np.log(Rt),3)))
31 )
32
33     # Convert from Kelvin to Celsius
34     TempC = TempK - 273.15
35
36     print("Sample:", i)
37     print("Voltage Value:", round(Vout,2))
38     print("Celsius Value:", round(TempC,1))
39     print("\n")
40     time.sleep(1)
41     i = i+1
42
43 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:

<sup>1</sup> 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

```
1 import nidaqmx
2
3 task = nidaqmx.Task()
4
5 task.ai_channels.add_ai_thrmcp_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.

# **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 Approximation, 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>

### 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.

```
1 import matplotlib.pyplot as plt    # MATLAB plotting functions
2 from control.matlab import *    # MATLAB-like functions
3
4 # Parameters defining the system
5 m = 250.0                      # system mass
6 k = 40.0                        # spring constant
7 b = 60.0                        # damping constant
8
9 # System matrices
10 A = [[0, 1.], [-k/m, -b/m]]
11 B = [[0], [1/m]]
12 C = [[1., 0]]
13 sys = ss(A, B, C, 0)
14
15 # Step response for the system
16 plt.figure(1)
17 yout, T = step(sys)
18 plt.plot(T.T, yout.T)
19
20 # Bode plot for the system
21 plt.figure(2)
22 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

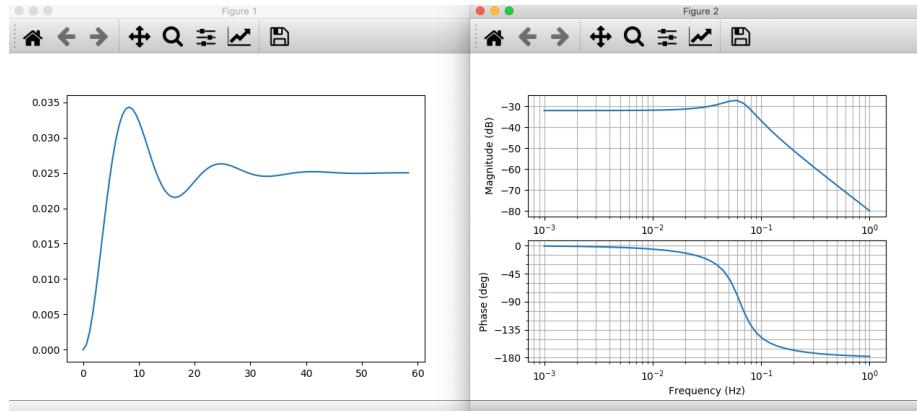


Figure 29.1: MATLAB compatibility module Example

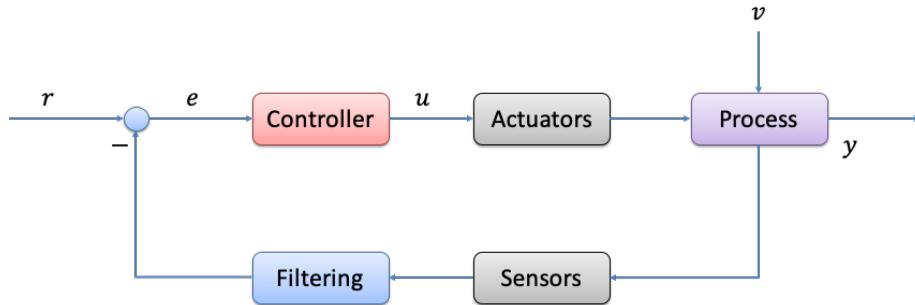


Figure 29.2: Control System

- 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:

$$H(s) = \frac{y(s)}{u(s)} \quad (29.1)$$

#### Example 29.2.1. Basic Transfer Function in Python

Lets define the following transfer function in Python:

$$H(s) = \frac{3}{4s + 1} \quad (29.2)$$

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 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:

$$\dot{x} = Ax + Bu \quad (29.3)$$

$$y = Cx + Du \quad (29.4)$$

Where A, B, C and D are matrices.

#### Example 29.2.2. State Space Model

Given the following system):

$$\dot{x}_1 = x_2 \quad (29.5)$$

$$2\dot{x}_1 = -2x_1 - 6x_2 + 4u_1 + 8u_2 \quad (29.6)$$

$$y = 5x_1 + 6x_2 + 7u_1 \quad (29.7)$$

This gives the following state space model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad (29.8)$$

$$y = \begin{bmatrix} 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} 7 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad (29.9)$$

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)
10
11 # Step response for the system
12 t, y = control.step_response(ssmodel)
13 plt.plot(t, y)
14
15
16 # Convert State space model to Transfer Function(s)
17 H = control.ss2tf(ssmodel)

```

Listing 29.3: State space model

Here we have also used the ss2tf() function. This function can convert a state space model to transfer function(s).

We have also the tf2ss() function that convert a transfer function to a 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.

The PID Algorithm (29.10):

$$u(t) = K_p e + \frac{K_p}{T_i} \int_0^t e d\tau + K_p T_d \dot{e} \quad (29.10)$$

Where  $u$  is the controller output and  $e$  is the control error:

$$e(t) = r(t) - y(t) \quad (29.11)$$

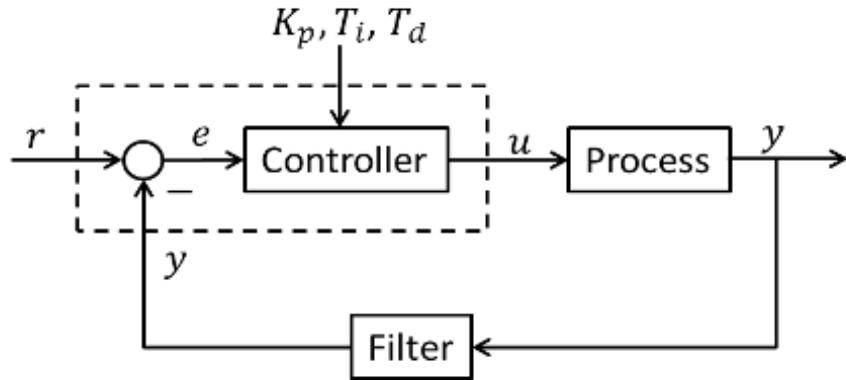


Figure 29.3: Control System using PID Controller

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):

$$H_{pid}(s) = \frac{u(s)}{e(s)} = K_p + \frac{K_p}{T_i s} + K_p T_d s \quad (29.12)$$

Or like this:

$$H_{pid}(s) = \frac{u(s)}{e(s)} = \frac{K_p(T_d T_i s^2 + T_i s + 1)}{T_i s} \quad (29.13)$$

### 29.3.1 PI Control

Very often we only need to use the PI Algorithm (29.14):

$$u(t) = K_p e + \frac{K_p}{T_i} \int_0^t e d\tau \quad (29.14)$$

The transfer function for the PI controller is:

$$H_{pi}(s) = \frac{u(s)}{e(s)} = \frac{K_p(T_i s + 1)}{T_i s} \quad (29.15)$$

# 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:

$$H(s) = \frac{y(s)}{u(s)} \quad (30.1)$$

Where  $y$  is the output and  $u$  is the input. A general transfer function can be written on the following general form:

$$H(s) = \frac{\text{numerator}(s)}{\text{denominator}(s)} \quad (30.2)$$

or to be more specific:

$$H(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0} \quad (30.3)$$

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:

$$H(s) = \frac{3}{4s + 1} \quad (30.4)$$

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:

$$H(s) = \frac{3s + 3}{4s^2 + 5s + 6} \quad (30.5)$$

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:

$$H(s) = \frac{x(s)}{u(s)} = \frac{K}{Ts + 1} \quad (30.6)$$

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):

$$\dot{x} = \frac{1}{T}(-x + Ku) \quad (30.7)$$

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.

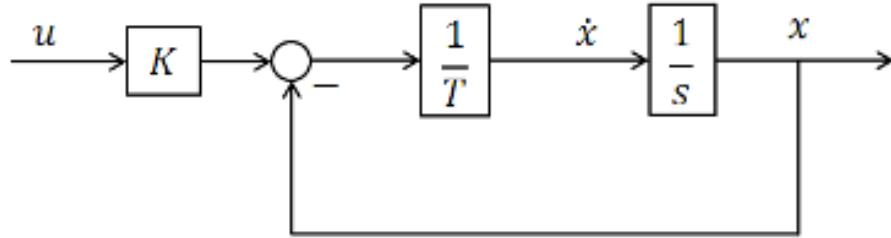


Figure 30.1: Block Diagram 1.order System

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:

$$H(s) = \frac{3}{4s + 1} \quad (30.8)$$

The Python code becomes:

```

1 import numpy as np
2 import control
3
4 K = 3
5 T = 4
6
7 num = np.array ([K])
8 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.

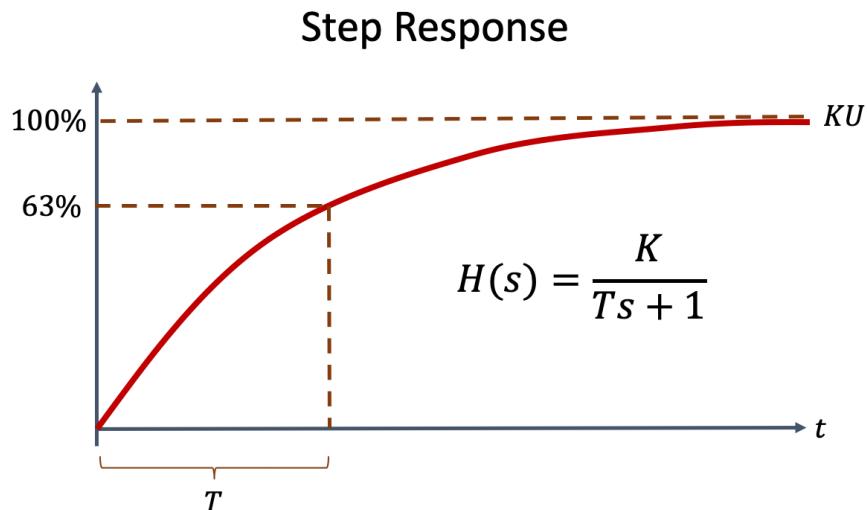


Figure 30.2: 1.order Process

Given the 1.order transfer function:

$$H(s) = \frac{y(s)}{u(s)} = \frac{K}{Ts + 1} \quad (30.9)$$

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:

$$\frac{1}{s} \Leftrightarrow 1 \quad (30.10)$$

Or more general:

$$\frac{U}{s} \Leftrightarrow U \quad (30.11)$$

Where  $U$  is the step size.

The step response of an integrator then becomes:

$$y(s) = H(s)u(s) \quad (30.12)$$

Where

$$u(s) = \frac{U}{s} \quad (30.13)$$

This gives:

$$y(s) = H(s)\frac{U}{s} = \frac{K}{Ts+1}\frac{U}{s} \quad (30.14)$$

We use the following Laplace Transformation pair in order to find  $y(t)$ :

$$\frac{k}{(Ts+1)s} \Leftrightarrow k(1 - e^{-t/T}) \quad (30.15)$$

This gives the following:

$$y(t) = KU(1 - e^{-t/T}) \quad (30.16)$$

In Python we can use the `step_response` function from the Python Control Systems Library.

Python code:

```
1 >>> 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:

$$H(s) = \frac{3}{4s + 1} \quad (30.17)$$

The Python code for plotting the step response:

```

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 H = control.tf(num, den)
12 print('H(s) =', H)
13
14 t, y = control.step_response(H)
15 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:

$$H(s) = \frac{3}{4s + 1} \quad (30.18)$$

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:

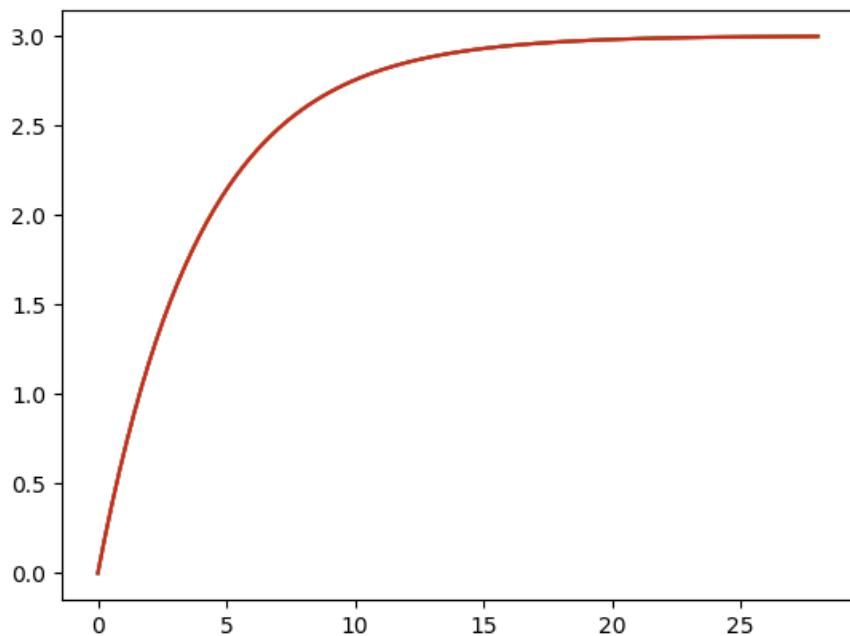


Figure 30.3: Step Response Example

```

1 import matplotlib.pyplot as plt
2 import control
3
4 s = control.TransferFunction.s
5
6 H = (3)/(4*s + 1)
7
8 print ('H(s) =', H)
9
10 t, y = control.step_response(H)
11
12 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 Delay

A 1.order transfer function with time-delay may be written as:

$$H(s) = \frac{K}{Ts + 1} e^{-\tau s} \quad (30.19)$$

In the time domain we get the following differential equation (using Inverse Laplace):

$$\dot{x} = \frac{1}{T}(-x + Ku(t - \tau)) \quad (30.20)$$

We can draw the following block diagram of the system (30.4).

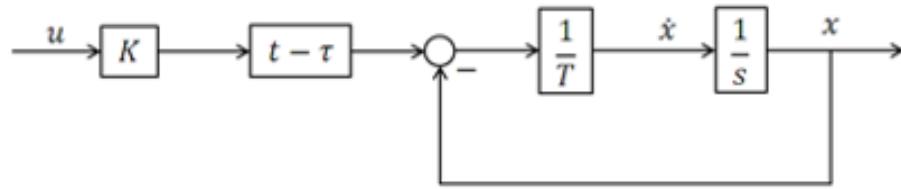


Figure 30.4: Block Diagram 1.order with Time Delay

Step response for a 1.order process with time delay, see Figure 34.5.

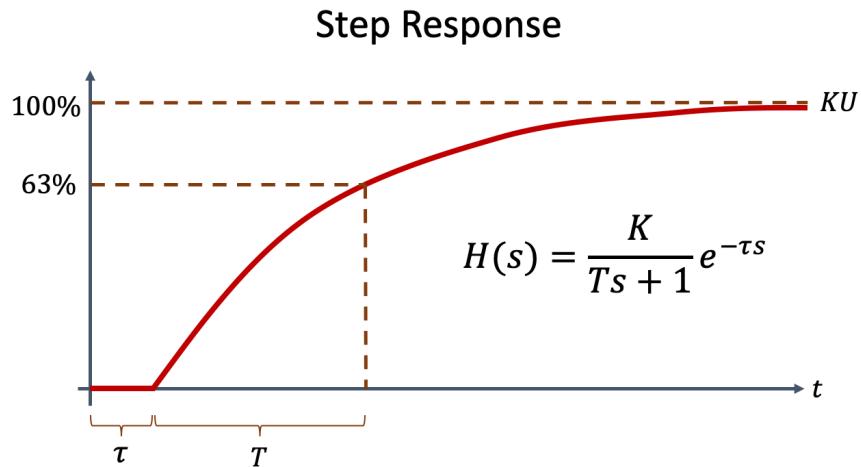


Figure 30.5: 1.order Process with Time delay

**Example 30.2.1.** 1.order Transfer Function with Time Delay using Pade Approximation

Given the following transfer function with time delay:

$$H(s) = \frac{3}{4s+1} e^{-2s} \quad (30.21)$$

The Python code for plotting the step response:

```

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):

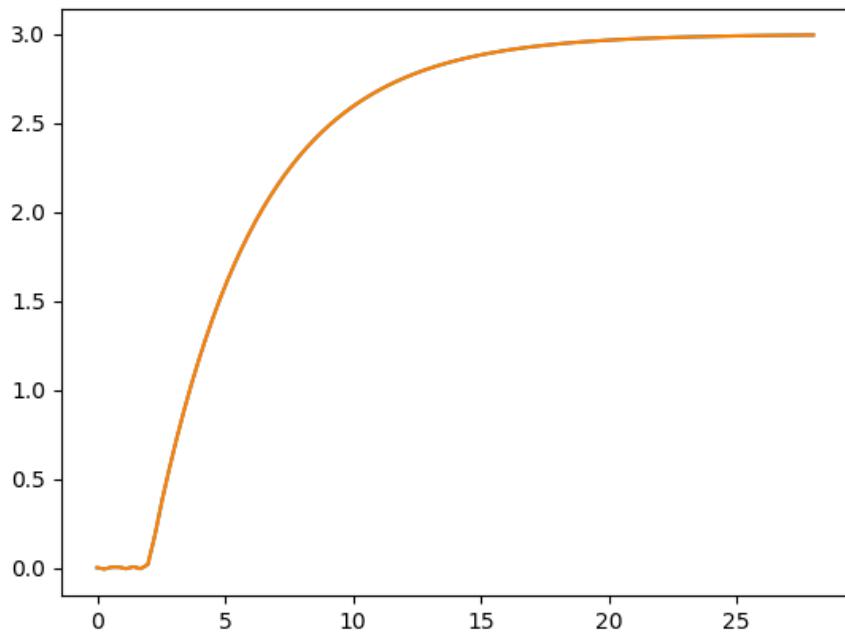


Figure 30.6: Step Response 1.order with Time Delay using Pade

$$H(s) = \frac{y(s)}{u(s)} = \frac{K}{s} \quad (30.22)$$

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:

$$\frac{1}{s} \Leftrightarrow 1 \quad (30.23)$$

The step response of an integrator then becomes:

$$y(s) = H(s)u(s) = \frac{K}{s} \frac{U}{s} = KU \frac{1}{s^2} \quad (30.24)$$

We use the following Laplace Transformation pair in order to find  $y(t)$ :

$$\frac{1}{s^2} \Leftrightarrow t \quad (30.25)$$

Then we get the following:

$$y(t) = KUt \quad (30.26)$$

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):

$$H(s) = \frac{y(s)}{u(s)} = \frac{K}{s} \quad (30.27)$$

We will find poles, perform step response, etc. in Python for this system.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import control
4
5 K = 3
6 num = np.array([K])
7 den = np.array([1, 0])
8 H = control.tf(num, den)
9 print ('H(s) =', H)
10 p = control.pole(H)
11
12
13
14 t, y = control.step_response(H)
15 plt.plot(t,y)
16 plt.title('Step Response')
17 plt.xlabel('t[s]')
18 plt.ylabel('y(t)')
19 plt.grid()
20 plt.axis([0, 7, 0, 22])
21 plt.show()
```

```

22 control.pzmap(H)
23 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):

$$H(s) = \frac{K}{as^2 + bs + c} \quad (30.28)$$

We can also define a 2.order transer function like this (30.29):

$$H(s) = \frac{K}{(\frac{s}{\omega_0})^2 + 2\zeta\frac{s}{\omega_0} + 1} \quad (30.29)$$

Where

K is the gain

$\zeta$  is the relative damping factor

$\omega_0[\text{rad/s}]$  is the undamped resonance frequency

2.order system - special case:

$$H(s) = \frac{K}{(T_1 s + 1)(T_2 s + 1)} \quad (30.30)$$

We see that this system can be considered as two 1.order systems in series:

$$H(s) = \frac{K}{(T_1 s + 1)} \frac{1}{(T_2 s + 1)} = \frac{K}{(T_1 s + 1)(T_2 s + 1)} \quad (30.31)$$

## 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:

$$H(s) = \frac{3}{4s + 1} e^{-2s} \quad (30.32)$$



Figure 30.7: Serial Block Diagram

$$H(s) = \frac{5}{2s+1} e^{-2s} \quad (30.33)$$

Python code:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import control
4
5 K = 3
6 T = 4
7 num = np.array ([K])
8 den = np.array ([T , 1])
9 H1 = control.tf(num,den)
10
11 K = 5
12 T = 2
13 num = np.array ([K])
14 den = np.array ([T , 1])
15 H2 = control.tf(num,den)
16
17 H = control.series(H1, H2)
18
19 print ('H(s) =', H)
20
21 t , y = control.step_response(H)
22 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

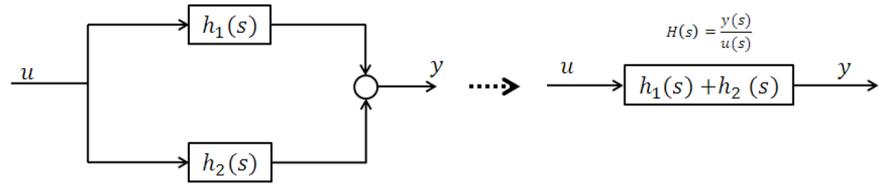


Figure 30.8: Parallel Block Diagram

Given the following transfer functions:

$$H(s) = \frac{3}{4s+1} e^{-2s} \quad (30.34)$$

$$H(s) = \frac{5}{2s+1} e^{-2s} \quad (30.35)$$

Python code:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import control
4
5 K = 3
6 T = 4
7 num = np.array ([K])
8 den = np.array ([T , 1])
9 H1 = control.tf(num,den)
10
11 K = 5
12 T = 2
13 num = np.array ([K])
14 den = np.array ([T , 1])
15 H2 = control.tf(num,den)
16
17 H = control.parallel(H1, H2)
18
19 print ('H(s) =', H)
20
21 t, y = control.step_response(H)
22 plt.plot(t,y)

```

Listing 30.10: Parallel Block Diagram Example

[End of Example]

### 30.5.3 Feedback Block Diagrams

Feedback:

Figure 30.7 shows a feedback block diagram.

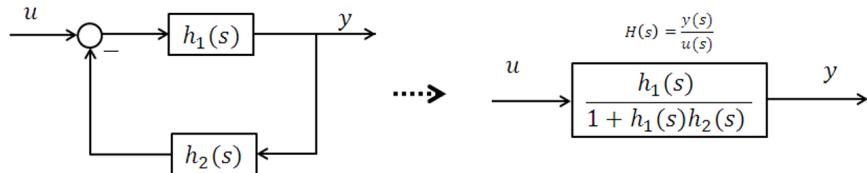


Figure 30.9: Feedback Block Diagram

**Example 30.5.3.** Feedback Block Diagram

Given the following transfer functions:

$$H(s) = \frac{3}{4s+1}e^{-2s} \quad (30.36)$$

$$H(s) = \frac{5}{2s+1}e^{-2s} \quad (30.37)$$

Python code:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import control
4
5 K = 3
6 T = 4
7 num = np.array ([K])
8 den = np.array ([T , 1])
9 H1 = control.tf(num,den)
10
11 K = 5
12 T = 2
13 num = np.array ([K])
14 den = np.array ([T , 1])
15 H2 = control.tf(num,den)
16
17 H = control.feedback(H1, H2)
18
19 print ('H(s) =', H)
20
21 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:

$$\dot{x}_1 = a_{11}x_1 + a_{21}x_2 + \dots + a_{n1}x_n + b_{11}u_1 + b_{21}u_2 + \dots + b_{n1}u_n$$

...

$$\dot{x}_n = a_{1n}x_1 + a_{2n}x_2 + \dots + a_{nn}x_n + b_{1n}u_1 + b_{2n}u_2 + \dots + b_{nn}u_n$$

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):

$$\dot{x}_1 = x_2 \tag{31.3}$$

$$2\dot{x}_1 = -2x_1 - 6x_2 + 4u_1 + 8u_2 \quad (31.4)$$

$$y = 5x_1 + 6x_2 + 7u_1 \quad (31.5)$$

This gives the following state space model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad (31.6)$$

$$y = [5 \ 6] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} [7 \ 0] \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad (31.7)$$

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)
10
11 # Step response for the system
12 t, y = control.step_response(ssmodel)
13 plt.plot(t, y)
14
15
16 # Convert State space model to Transfer Function(s)
17 H = control.ss2tf(ssmodel)

```

Listing 31.1: State space model

Here we have also used the `ss2tf()` function. This function can convert a state space model to transfer function(s).

We have also the `tf2ss()` function that convert a transfer function to a 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.

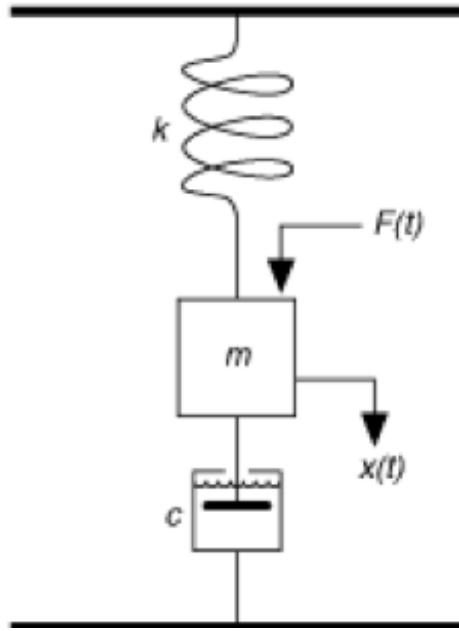


Figure 31.1: Mass-Spring-Damper System

The system can be described by the following equation:

$$F(t) - cx(t) - kx(t) = m\ddot{x}(t) \quad (31.8)$$

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).

$\ddot{x}(t)$  is the second derivative of the position, which equals the acceleration of the object (m).

The state space model can be stated like this:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u \quad (31.9)$$

$$y = [1 \ 0] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad (31.10)$$

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
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 ( $\phi$ ).

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:

$$u(t) = U \sin(\omega t) \quad (32.1)$$

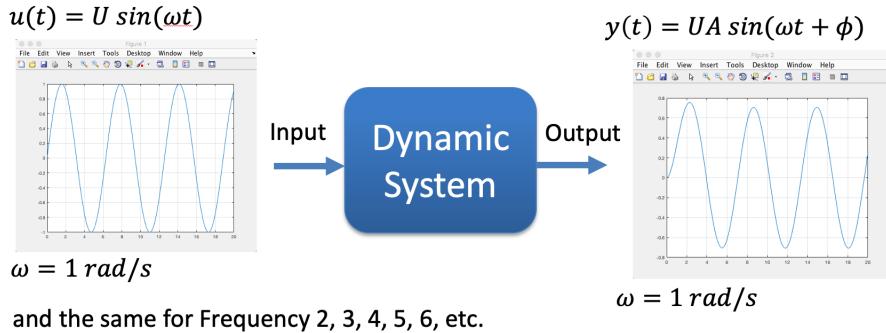


Figure 32.1: Frequency Response Definition

The steady-state output signal will be:

$$y(t) = UA \sin(\omega t + \phi) \quad (32.2)$$

Where  $A = \frac{Y}{U}$  is the ratio between the amplitudes of the output signal and the input signal (in steady-state).

$A$  and  $\phi$  is a function of the frequency  $\omega$  so we may write  $A = A(\omega)$  and  $\phi = \phi(\omega)$

For a transfer function:

$$H(S) = \frac{y(s)}{u(s)} \quad (32.3)$$

We have that:

$$H(j\omega) = |H(j\omega)| e^{j\angle H(j\omega)} \quad (32.4)$$

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

diagram,  $\phi(\omega)$ .

This is illustrated in Figure 32.2.

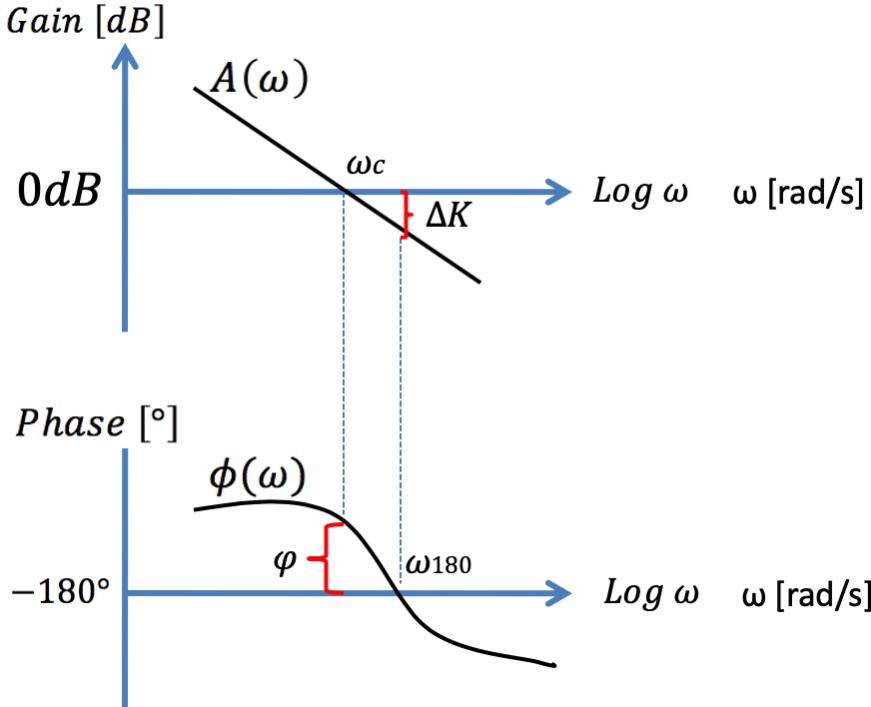


Figure 32.2: Bode Diagram

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:

$$A(\omega) = |H(j\omega)| \quad (32.5)$$

The Phase function:

$$\phi(\omega) = \angle H(j\omega) \quad (32.6)$$

The  $A(\omega)$  axis is in decibel (dB), where the decibel value of x is calculated as:

$$x[dB] = 20\log_{10}(x) \quad (32.7)$$

The  $\phi(\omega)$  axis is in degrees (not radians!)

### Example 32.2.1. Frequency Response - 1.order

Given the following transfer function:

$$H(s) = \frac{3}{4s + 1} \quad (32.8)$$

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 print ('H(s) =', H)
9
10 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:

```

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 print ('H(s) =', H)
9
10 wlist = [0.01, 0.1, 1, 2, 3, 5, 10, 100]
11 [mag, phase, w] = control.bode(H, wlist, dB=True)
12
13 # Convert to Decibel
14 magdb = 20 * np.log10(mag);
15 #magdb = control.mag2db(mag) Alternative solution: This is a
16 # premade function implementing the formula above
16 print(magdb)

```

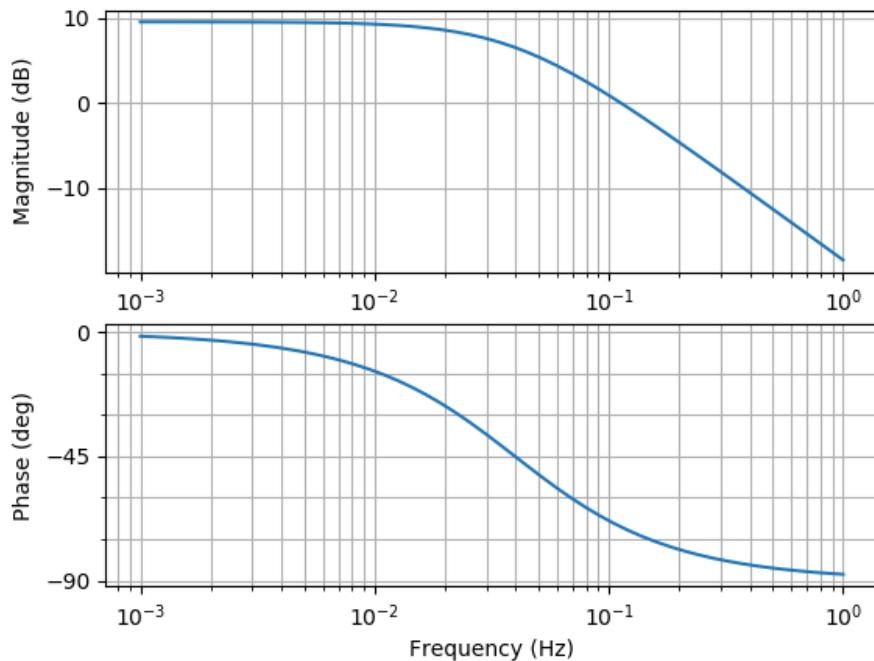


Figure 32.3: Frequency Response 1.order

```

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:

$$H(s) = \frac{3}{4s^2 + 5s + 6} \quad (32.9)$$

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)

```

```

9
10 control.bode(H, dB=True)

```

Listing 32.3: Frequency Response - 2.order

This gives the the plot shown in Figure 32.4.

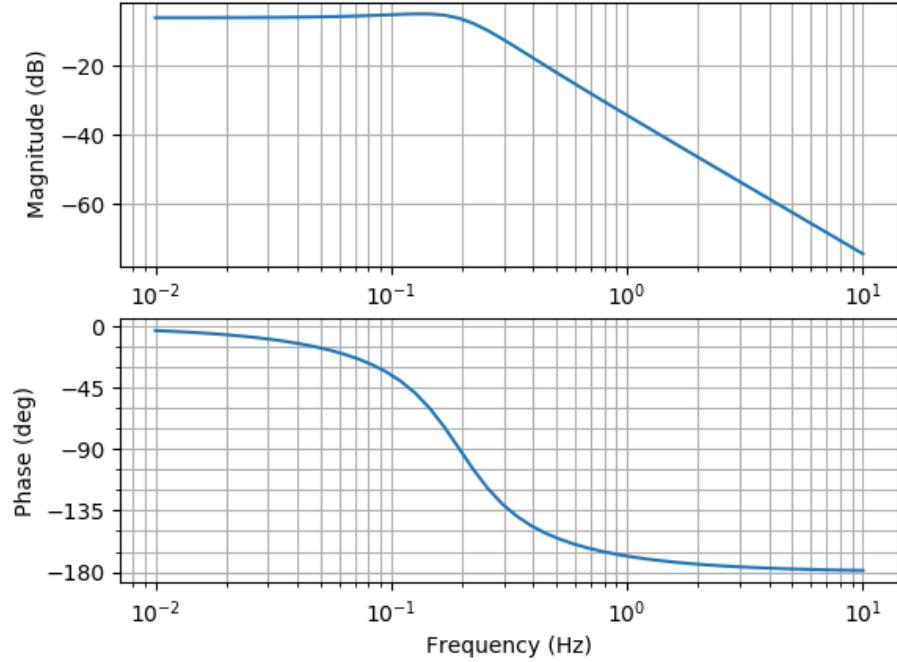


Figure 32.4: Frequency Response 2.order

[End of Example]

### Example 32.2.3. Frequency Response Example

Given the following transfer function:

$$H(s) = \frac{3s + 3}{4s^2 + 5s + 6} \quad (32.10)$$

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.

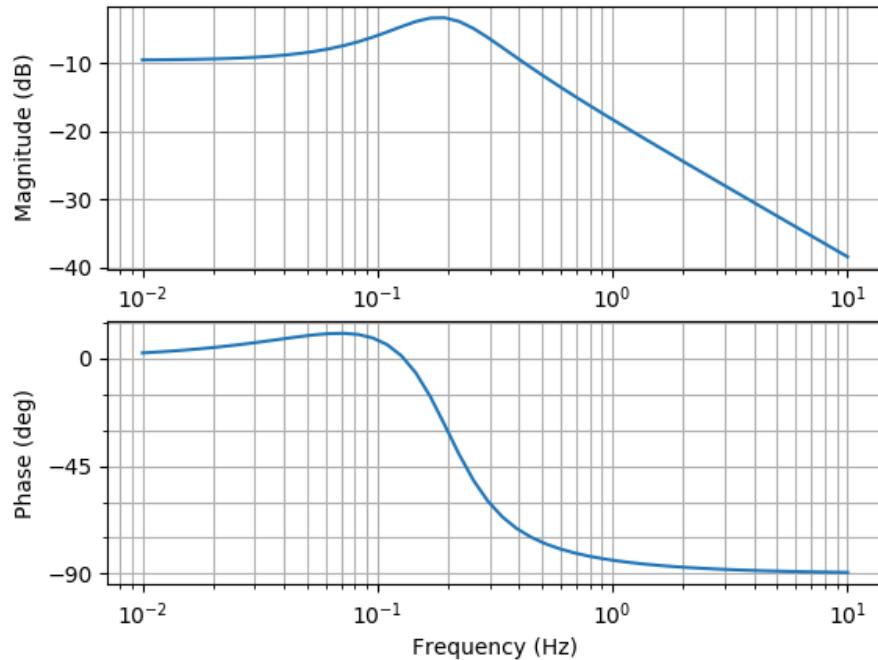


Figure 32.5: Frequency Response

[End of Example]

# Chapter 33

## Stability Analysis

### 33.1 Introduction

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>

A dynamic system has one of the following stability properties:

- Asymptotically stable system
- Marginally stable system
- Unstable system

#### 33.1.1 Asymptotically Stable System

An asymptotically stable system is defined by:

$$\lim_{t \rightarrow \infty} h(t) = k \quad (33.1)$$

Figure 33.1 illustrates an asymptotically stable system

#### 33.1.2 Marginally Stable System

A marginally stable system is defined by;

$$0 < \lim_{t \rightarrow \infty} h(t) < \infty \quad (33.2)$$

### Asymptotically stable system

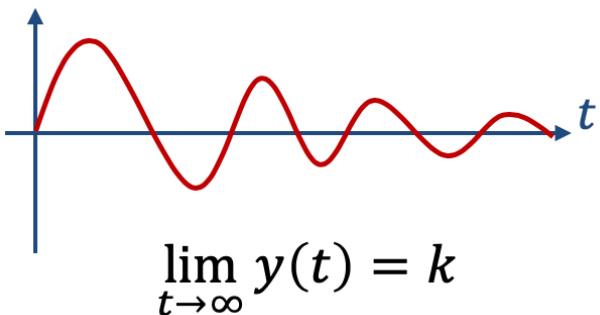


Figure 33.1: Asymptotically stable system

Figure 33.2 illustrates a marginally stable system.

### Marginally stable system

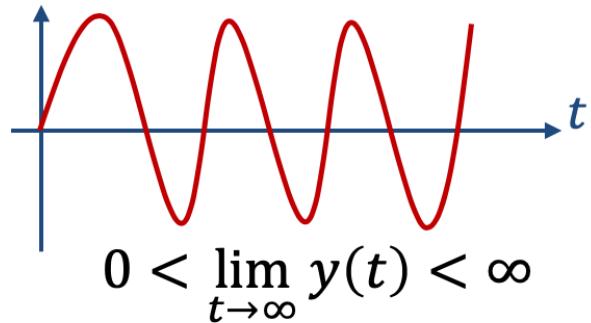


Figure 33.2: Marginally Stable System

### 33.1.3 Unstable System

An unstable system is defined by:

$$\lim_{t \rightarrow \infty} h(t) = \infty \quad (33.3)$$

Figure 33.3 illustrates an unstable system.

## Unstable system

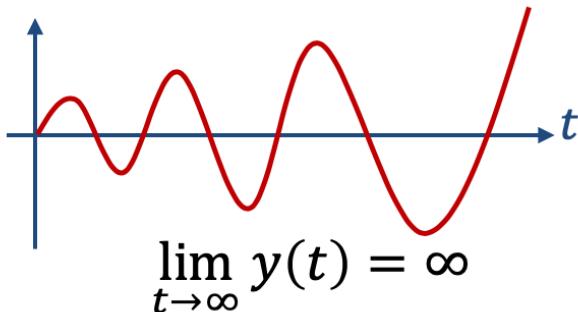


Figure 33.3: Unstable System

How do we figure out that the feedback system is stable before we test it on the real system?

Here are 3 different methods:

- Poles
- Frequency Response/Bode
- Simulations (Step Response)

We will do all these things using Python.

Figure 33.4 illustrates the 3 different methods mentioned.

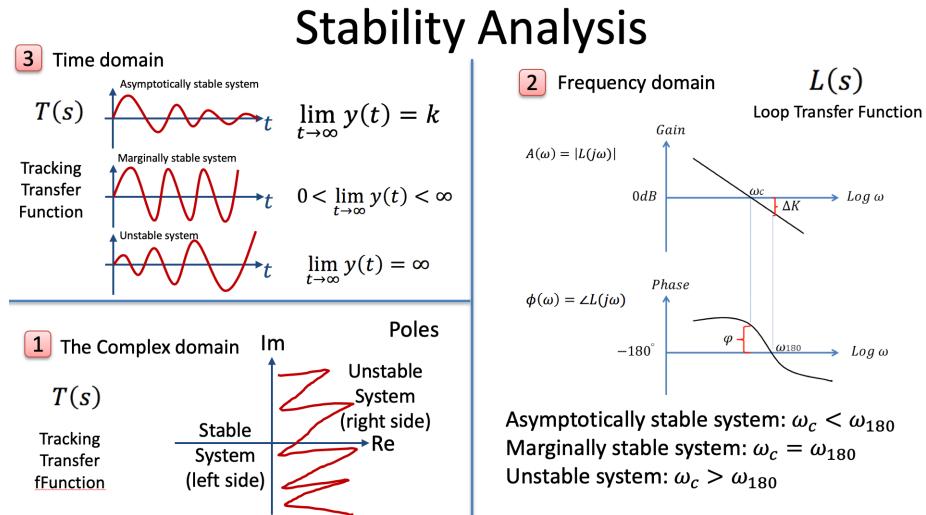


Figure 33.4: Stability Analysis

## 33.2 Poles

The poles are important when analysing the stability of a system. Figure 33.5 gives an overview of the poles impact on the stability of a system.

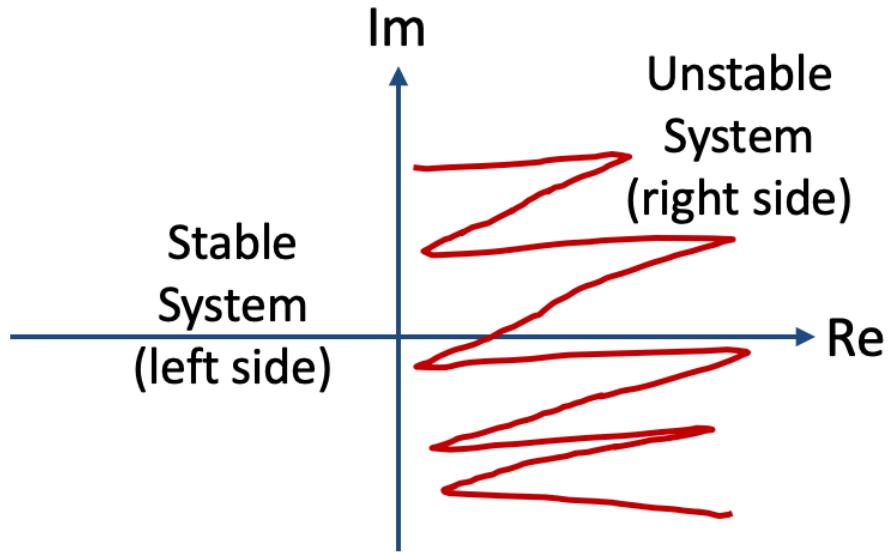


Figure 33.5: Poles in the Complex Domain

Figure 33.6 illustrates the 3 different methods mentioned.

Figure 33.7 illustrates the 3 different methods mentioned.

### 33.2.1 Asymptotically Stable System

Each of the poles of the transfer function lies strictly in the left half plane (has strictly negative real part).

Figure 33.8 illustrates this.

### 33.2.2 Marginally Stable System

One or more poles lies on the imaginary axis (have real part equal to zero), and all these poles are distinct. Besides, no poles lie in the right half plane.

Figure 33.9 illustrates this.

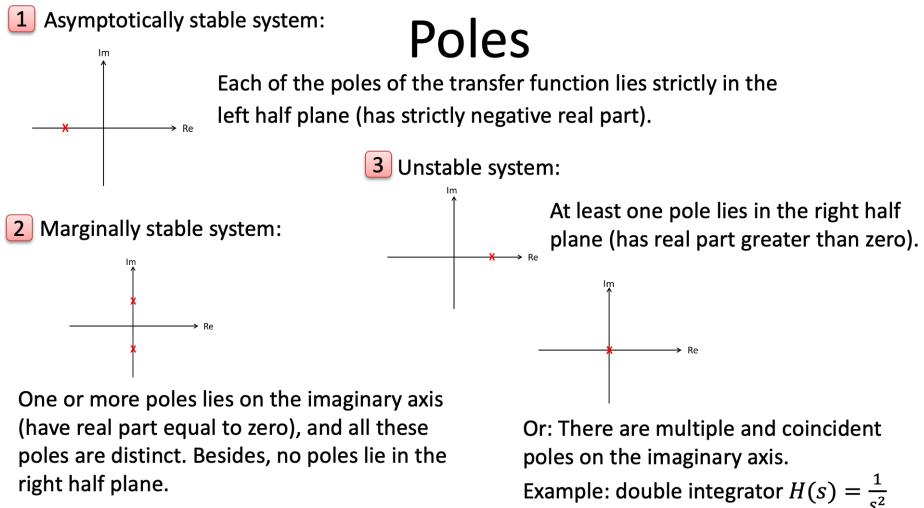


Figure 33.6: Stability Analysis

### 33.2.3 Unstable System

At least one pole lies in the right half plane (has real part greater than zero). Or there are multiple and coincident poles on the imaginary axis (Example: double integrator  $H(s) = \frac{1}{s^2}$  ).

Figure 33.10 illustrates this.

## 33.3 Feedback Systems

Here are some important transfer functions to determine the stability of a feedback system. Figure 33.11 shows a typical feedback system.

### 33.3.1 Loop Transfer Function

The Loop transfer function  $L(s)$  (Norwegian: “Sløyfetransferfunksjonen”) is defined as follows (33.4):

$$L(s) = H_c(s)H_p(s)H_m(s) \quad (33.4)$$

Where

$H_p(s)$  is the transfer function for the process

$H_m(s)$  is the transfer function for the measurement (sensor)

$H_c(s)$  is the transfer function for the controller

Note! Another notation for  $L$  is  $H_0$

# Stability Analysis

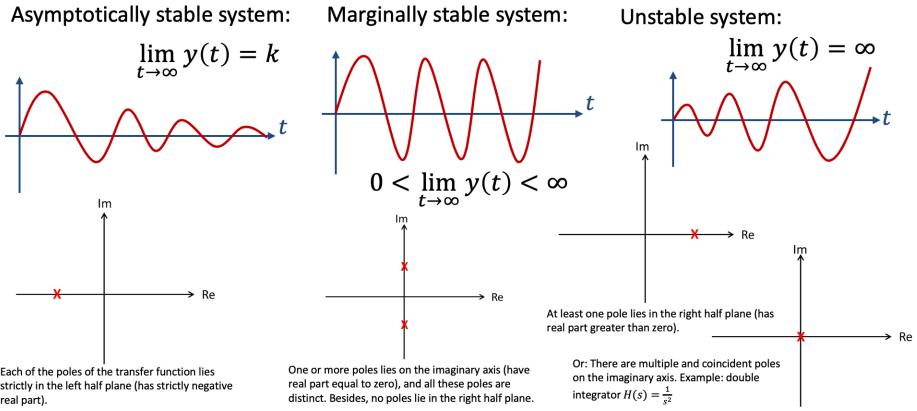


Figure 33.7: Stability Analysis

### 33.3.2 Tracking Transfer Function

The Tracking transfer function  $T(s)$  (Norwegian: “Følgeforholdet”) is defined as follows (33.21):

$$T(s) = \frac{y(s)}{r(s)} = \frac{H_c(s)H_p(s)H_m(s)}{1 + H_c(s)H_p(s)H_m(s)} = \frac{L(s)}{1 + L(S)} = 1 - S(s) \quad (33.5)$$

The Tracking Property (Norwegian: “følgeegenskaper”) is good if the tracking function  $T$  has value equal to or close to 1:

$$|T| \approx 1 \quad (33.6)$$

### 33.3.3 Sensitivity Transfer Function

The Sensitivity transfer function  $S(s)$  (Norwegian: “Sensitivitetsfunksjonen/avviks-forholdet”) is defined as follows:

Given the following (34.35):

$$S(s) = \frac{e(s)}{r(s)} = \frac{1}{1 + L(s)} = 1 - T(s) \quad (33.7)$$

## Asymptotically Stable System

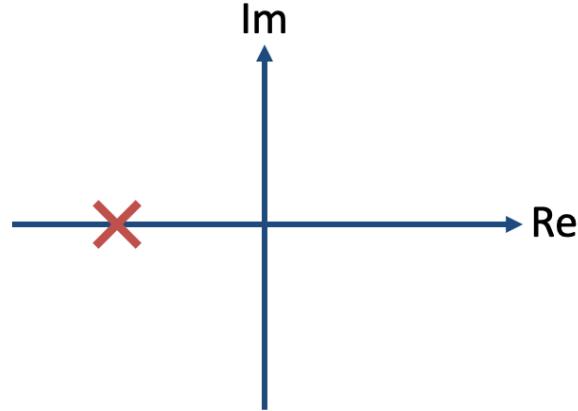


Figure 33.8: Poles Asymptotically Stable System

The Compensation Property is good if the sensitivity function  $S$  has a small value close to zero:

$$|S| \approx 0 \text{ or } |S| \ll 1 \quad (33.8)$$

Note!

$$T(s) + S(s) = \frac{L(s)}{1 + L(s)} = \frac{1}{1 + L(s)} \equiv 1 \quad (33.9)$$

### 33.3.4 Characteristic Polynomial

We have that:

$$L(s) = \frac{n_L(s)}{d_L(s)} \quad (33.10)$$

And:

$$T(s) = \frac{y(s)}{r(s)} = \frac{L(s)}{1 + L(s)} = \frac{\frac{n_L(s)}{d_L(s)}}{1 + \frac{n_L(s)}{d_L(s)}} = \frac{n_L(s)}{d_L(s) + n_L(s)} \quad (33.11)$$

## Marginally Stable System

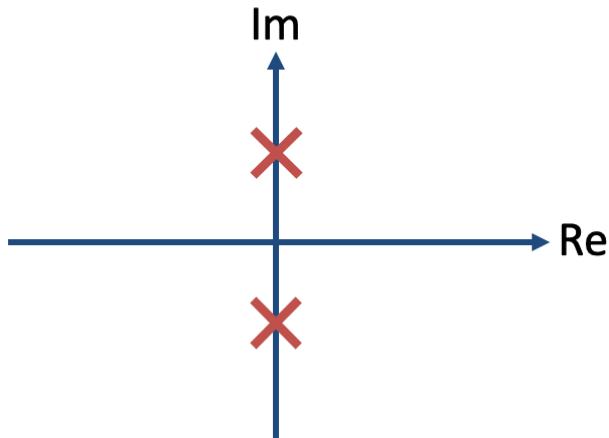


Figure 33.9: Poles Marginally Stable System

Where  $n_L(s)$  and  $d_L(s)$  numerator and the denominator of the Loop transfer function  $L(s)$ .

The Characteristic Polynomial for the control system then becomes:

$$a(s) = d_L(s) + n_L(s) \quad (33.12)$$

## 33.4 Frequency Response and stability Analysis

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 diagram,  $\phi(\omega)$ .

This is illustrated in Figure 33.12.

The frequencies  $\omega_c$  and  $\omega_{180}$  are called the crossover-frequencies (or “kryssfrekvens” in Norwegian).

$\Delta K$  is the gain margin (GM) of the system (or “Forsterkningsmargin” in Norwegian). The gain margin illustrates how much the loop gain can increase before the system becomes unstable.

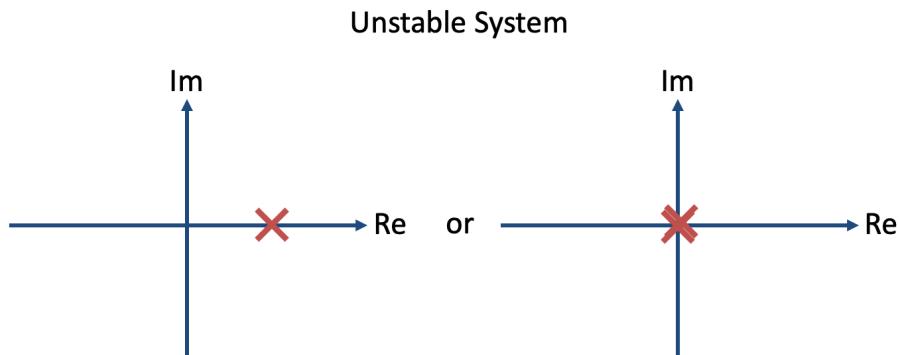


Figure 33.10: Poles Unstable System

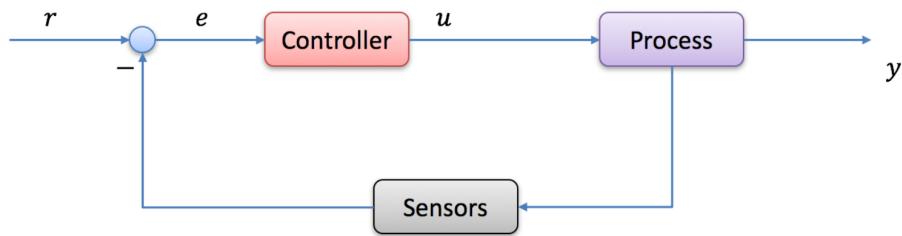


Figure 33.11: Feedback System

$\phi$  is the phase margin (PM) of the system (or “Fasemmargin” in Norwegian). The phase margin illustrates much phase shift the system can tolerate before it becomes unstable.

The definitions are as follows:

Gain Crossover-frequency -  $\omega_c$ :

$$|L(j\omega_c)| = 1 = 0dB \quad (33.13)$$

$\omega_c$  is phase margin frequency, in radians/second. A phase margin frequency indicates where the model magnitude crosses 0 decibels.

Phase Crossover-frequency -  $\omega_{180}$ :

$$\angle L(j\omega_{180}) = -180 degrees \quad (33.14)$$

$\omega_{180}$  is the gain margin frequency, in radians/second. A gain margin frequency indicates where the model phase crosses -180 degrees.

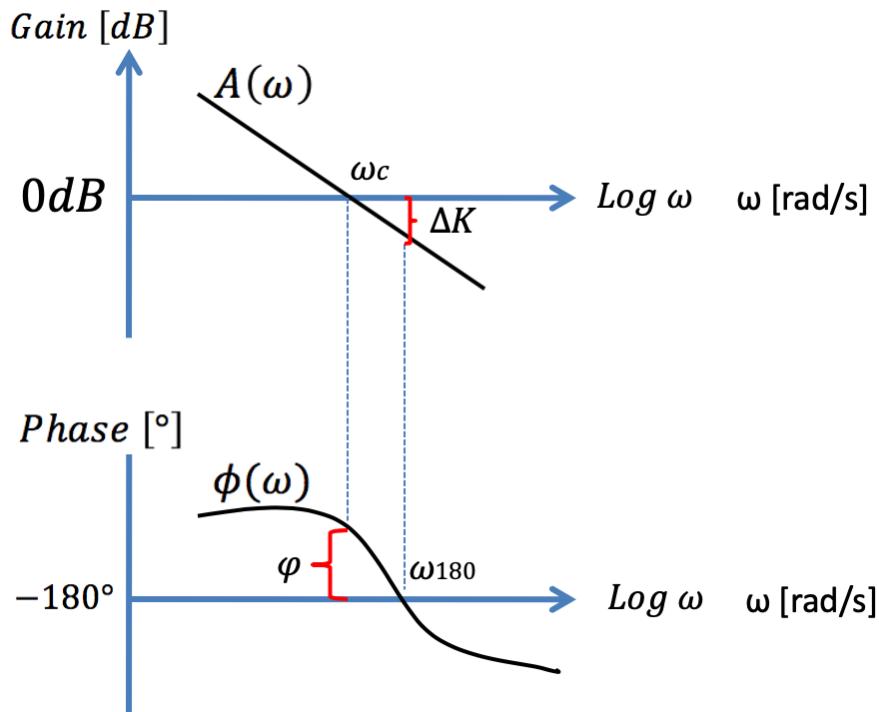


Figure 33.12: Bode Diagram used in Stability Analysis

Gain Margin - GM ( $\Delta K$ ):

$$GM[dB] = \Delta K = -|L(j\omega_{180})| [dB] \quad (33.15)$$

Phase margin PM ( $\phi$ ):

$$PM = \phi = 180 \text{ degrees} + \angle L(j\omega_c) \quad (33.16)$$

We have the following:

- Asymptotically stable system:  $\omega_c < \omega_{180}$
- Marginally stable system:  $\omega_c = \omega_{180}$
- Unstable system:  $\omega_c > \omega_{180}$

### 33.5 Examples

**Example 33.5.1.** Stability Analysis of Feedback System

Given the system as shown in Figure 33.13.

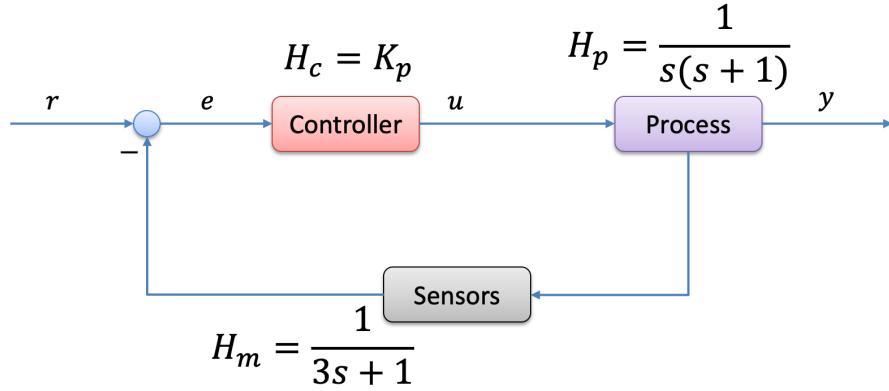


Figure 33.13: Feedback System Example

The system has the following transfer functions:

Transfer function for the Process:

$$H_p = \frac{1}{s(s+1)} = \frac{1}{s^2 + s} \quad (33.17)$$

Transfer function for the Sensor (or Measurement):

$$h_m = \frac{1}{3s+1} \quad (33.18)$$

Transfer function for the Controller:

$$H_c = K_p \quad (33.19)$$

We will use Python to find loop transfer function  $L(s)$ , the tracking transfer function  $T(s)$  and the sensitivity transfer function  $S(s)$ .

The Loop transfer function is defined as:

$$L(s) = H_c(s)H_p(s)H_m(s) \quad (33.20)$$

We will use the **series()** function in Python.

The Tracking transfer function  $T(s)$  is defined as follows (33.21):

$$T(s) = \frac{y(s)}{r(s)} = \frac{H_c(s)H_p(s)H_m(s)}{1 + H_c(s)H_p(s)H_m(s)} = \frac{L(s)}{1 + L(S)} \quad (33.21)$$

We will use **feedback()** function in Python.

The Sensitivity transfer function  $S(s)$  is defined as follows:

$$S(s) = \frac{e(s)}{r(s)} = \frac{1}{1 + L(s)} = 1 - T(s) \quad (33.22)$$

We will plot the Bode plot for the system using e.g., the `bode()` function in Python.

We will find the crossover-frequencies ( $\omega_{180}$ ,  $\omega_c$ ) and stability margins GM ( $A(\omega)$ ,  $PM(\phi(\omega))$ ) of the system ( $L(s)$ ) from the Bode plot.

We will also plot a Bode diagram where the crossover-frequencies, GM and PM are illustrated. Here we can use the `margin()` function in Python.

We will also use also the `margin()` function in order to find values for  $\omega_{180}, \omega_c, A(\omega), \phi(\omega)$  directly.

We will compare and discuss the results. How much can you increase  $K_p$  before the system becomes unstable?

Initial Python Code:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import control
4
5
6
7 # ----- Define Transfer Functions -----
8
9 # Transfer Function Process
10 num_p = np.array ([1])
11 den_p = np.array ([1 , 1, 0])
12
13 Hp = control.tf(num_p, den_p)

```

```

14 print ('Hp(s) =', Hp)
15
16
17 # Transfer Function Sensor
18 num_m = np.array ([1])
19 den_m = np.array ([3 , 1])
20
21 Hm = control.tf(num_m, den_m)
22 print ('Hm(s) =', Hm)
23
24
25 # Transfer Function Controller
26 Kp = 0.35
27
28 num_c = np.array ([Kp])
29 den_c = np.array ([1])
30
31 Hc = control.tf(num_c, den_c)
32 print ('Hc(s) =', Hc)
33
34
35 # The Loop Transfer function
36 L = control.series(Hc, Hp, Hm)
37 print ('L(s) =', L)
38
39 # Tracking transfer function
40 T = control.feedback(L,1)
41 print ('T(s) =', T)
42
43 # Sensitivity transfer function
44 S = 1 - T
45 print ('S(s) =', S)
46
47 # ----- Plotting -----
48
49
50 # Step Response Feedback System (Tracking System)
51 t, y = control.step_response(T)
52 plt.figure(1)
53 plt.plot(t,y)
54 plt.title("Step Response Feedback System T(s)")
55
56
57 # Bode Diagram
58 plt.figure(2)
59 control.bode(L, dB=True, deg=True)
60
61
62 # Bode Diagram with Stability Margins
63 plt.figure(3)
64 control.bode(L, dB=True, deg=True, margins=True)
65
66
67 # Poles and Zeros
68 control.pzmap(T)
69
70 p = control.pole(T)
71 z = control.zero(T)
72
73 print("poles = ", p)
74
75

```

```

76 # ----- Print Stability Analysis -----
77
78 # Calculating stability margins and crossover frequencies
79 gm , pm , w180 , wc = control.margin(L)
80
81 # Convert gm to Decibel
82 gmdb = 20 * np.log10(gm)
83
84 print("wc =", f'{wc:.2f}', "rad/s")
85 print("w180 =", f'{w180:.2f}', "rad/s")
86
87 print("GM =", f'{gm:.2f}')
88 print("GM =", f'{gmdb:.2f}', "dB")
89 print("PM =", f'{pm:.2f}', "deg")

```

Listing 33.1: Stability Analysis

[End of Example]

### Asymptotically Stable System

In the calculations and simulations  $K_p = 0.35$  has been used.  
This gives the the plots shown in Figure 33.14.

As you see from the plots in Figure 33.14 we have an asymptotically stable system

From the calculations we get the following stability parameters:

$w_c = 0.26 \text{ rad/s}$ ,  $w_{180} = 0.58 \text{ rad/s}$ ,  $GM = 3.81$ ,  $GM = 11.62 \text{ dB}$ ,  $PM = 36.69 \text{ deg}$

We see that  $\omega_c < \omega_{180}$  from the Bode plot.

We also see from the step response:

$$\lim_{t \rightarrow \infty} h(t) = 1 \quad (33.23)$$

Finally, the poles are located in the left half plane.

All these things shows that the system is asymptotically stable.

How much can you increase  $K_p$  before the system becomes unstable?

We know from above that  $GM(\Delta K)$  is:

$$\Delta K = 3.81 \quad (33.24)$$

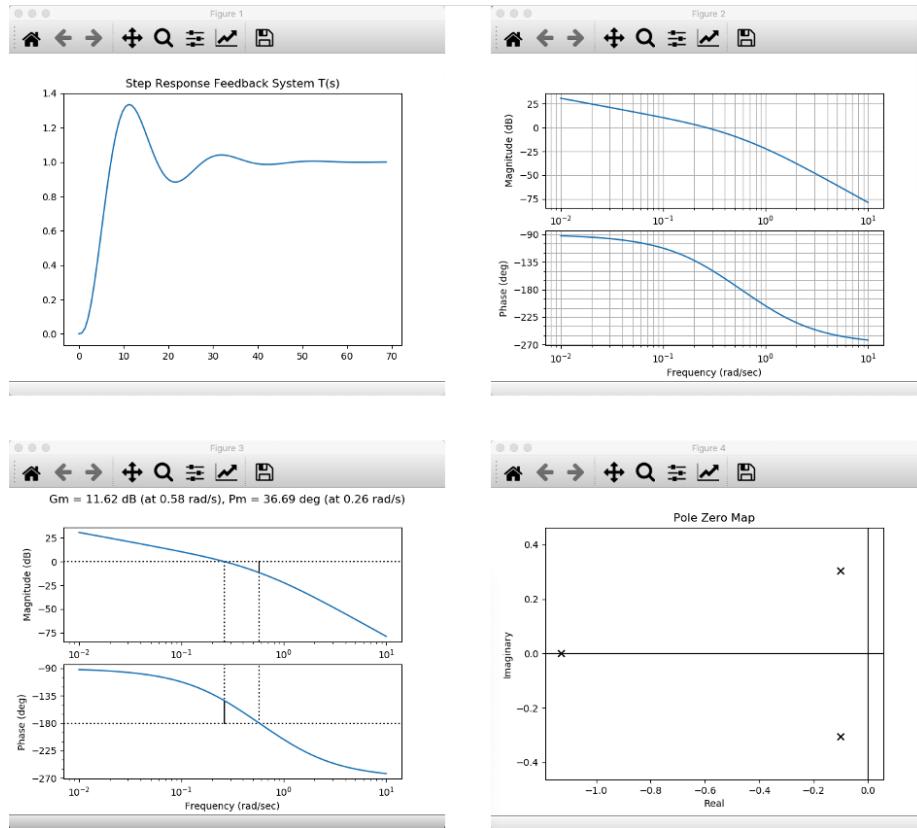


Figure 33.14: Stability Analysis Plots - Asymptotically Stable System

### Marginally Stable System

We find the  $K_p$  for a marginally stable system:

$$K_{pm} = 0.35 * \Delta K = 0.35 * 3.81 \approx 1.33 \quad (33.25)$$

When we use  $K_p = 1.33$  in the simulations, we get:

This gives the the plots shown in Figure 33.15.

As you see from the plots in Figure 33.15 we have a marginally stable system.

We see that  $\omega_c = \omega_{180}$  from the Bode plot.

We also see from the step response:

$$0 < \lim_{t \rightarrow \infty} h(t) < \infty \quad (33.26)$$

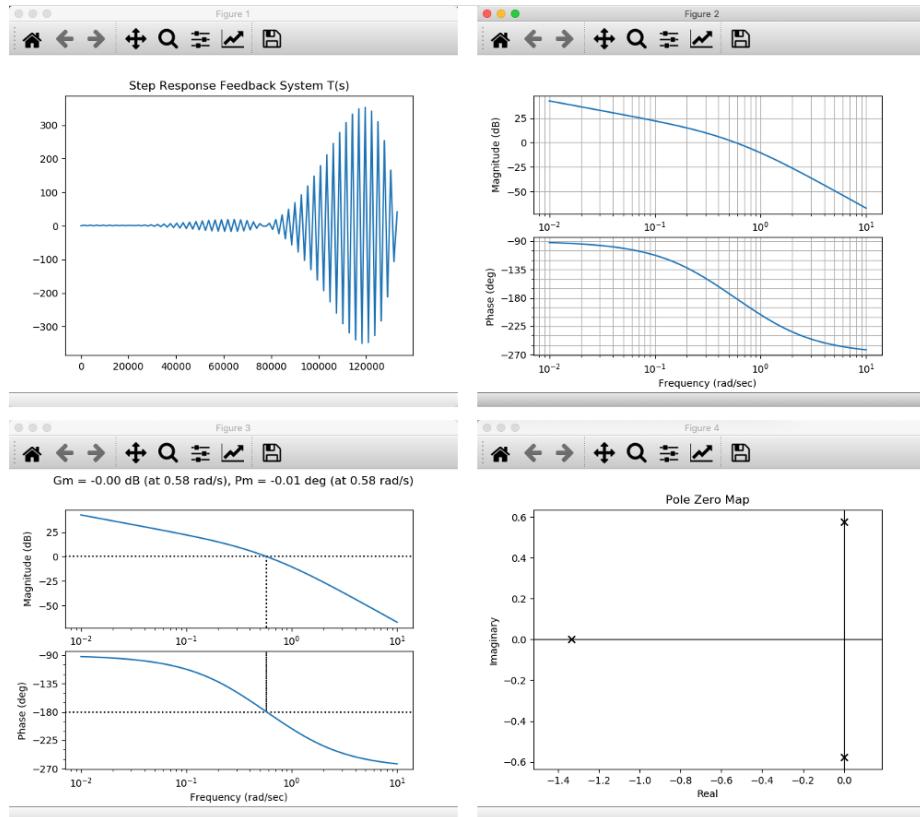


Figure 33.15: Stability Analysis Plots - Marginally Stable System

Finally, we have poles that lies on the imaginary axis (have real part equal to zero), and all these poles are distinct. Besides, no poles lie in the right half plane.

All these things shows that the system is marginally stable.

### Unstable System

All  $K_p$  values above this should give a unstable system. Lets try using  $K_p = 2$ .

This gives the the plots shown in Figure 33.16.

As you see from the plots in Figure 33.16 we have a marginally stable system.

We see that  $\omega_c > \omega_{180}$  from the Bode plot.

We also see from the step response:

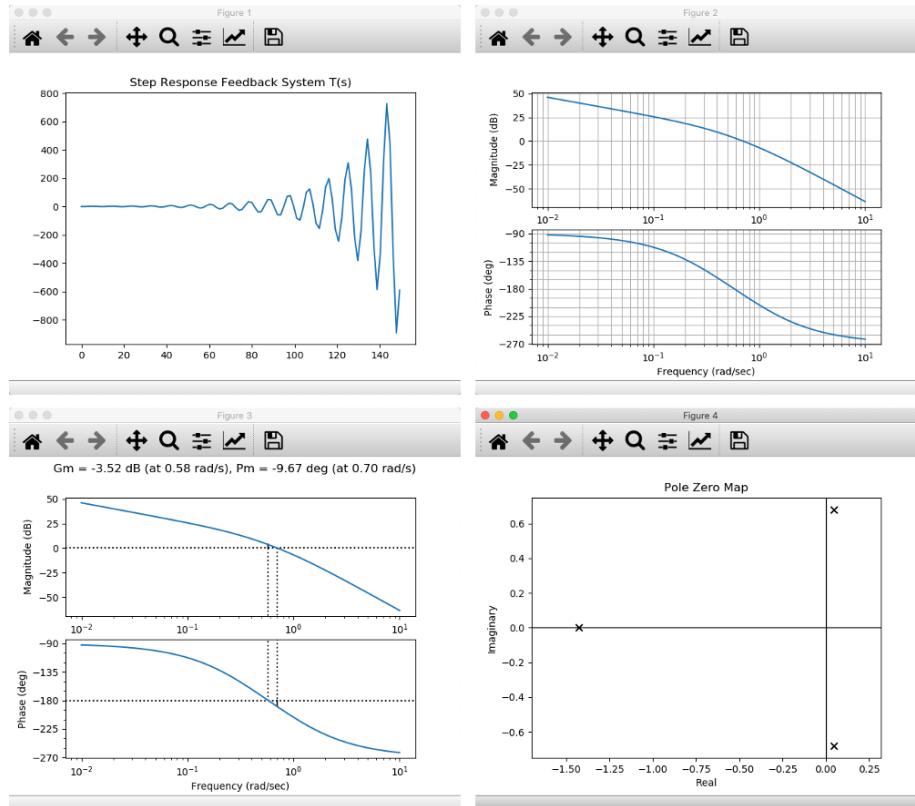


Figure 33.16: Stability Analysis Plots - Unstable System

$$\lim_{t \rightarrow \infty} h(t) = \infty \quad (33.27)$$

Finally, we have at least one pole lies in the right half plane (has real part greater than zero)

All these things shows that the system has become unstable.

So just by changing the value for  $K_p$  we can change behavior for the entire system, either asymptotically stable system, marginally stable system or an unstable system.

#### Example 33.5.2. PI Controller Feedback System

Given the system as shown in Figure 33.17.

The system has the following transfer functions:

Transfer function for the Process:

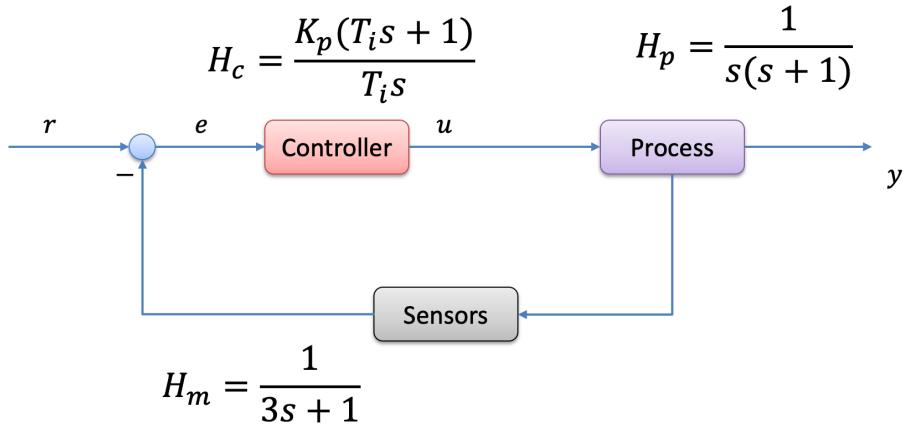


Figure 33.17: PI Controller Feedback System Example

$$H_p = \frac{1}{s(s + 1)} = \frac{1}{s^2 + s} \quad (33.28)$$

Transfer function for the Sensor (or Measurement):

$$h_m = \frac{1}{3s + 1} \quad (33.29)$$

Transfer function for the Controller:

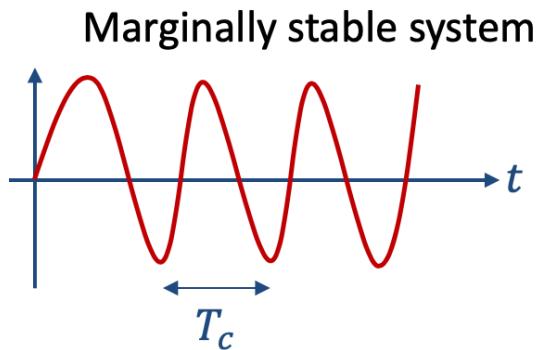
$$H_c = \frac{K_p(T_i s + 1)}{T_i s} \quad (33.30)$$

### Ziegler–Nichols Frequency Response Method

We will use the Ziegler–Nichols Frequency Response method in order to find proper PI parameters ( $K_p$  and  $T_i$ ) for this system.

The Ziegler–Nichols Frequency Response method can shortly be described like this:

Assume you use a P controller only, i.e.,  $T_i = \infty, T_d = 0$ . Then you need to find for which  $K_p$  the closed loop system is a marginally stable system ( $\omega_c = \omega_{180}$ ). This  $K_p$  is called  $K_c$  (critical gain). The  $T_c$  (critical period) can be found from the damped oscillations of the closed loop system, as shown in Figure 34.10.



$$\omega_c = \omega_{180}$$

$$0 < \lim_{t \rightarrow \infty} y(t) < \infty$$

$$T_c = \frac{2\pi}{\omega_{180}}$$

Figure 33.18: Damped Oscillations of the Closed Loop System

The critical period ( $T_c$ ) can be found (34.35):

$$T_c = \frac{2\pi}{\omega_{180}} \quad (33.31)$$

Then we can calculate the PI(D) parameters using the formulas given by Table 34.1.

Where

$K_c$  - Critical Gain  $T_c$  - Critical Period

#### PI Controller using Ziegler–Nichols Method

Ziegler–Nichols (PI Controller):

$$K_p = 0.45K_c \quad (33.32)$$

$$T_i = \frac{T_c}{1.2} \quad (33.33)$$

From Example we have:

$$K_c = 1.33 \quad (33.34)$$

$$\omega_{180} = 0.58 \text{ rad/s} \quad (33.35)$$

The critical period ( $T_c$ ) then becomes:

$$T_c = \frac{2\pi}{\omega_{180}} = \frac{2\pi}{0.58} \quad (33.36)$$

Finally, this gives the following PI Parameters:

$$K_p = 0.45K_c = 0.45 \times 1.33 \approx 0.6 \quad (33.37)$$

$$T_i = \frac{T_c}{1.2} = \frac{\frac{2\pi}{0.58}}{1.2} \approx 9s \quad (33.38)$$

Python code:

```

1 import math
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import control
5
6 # ----- Ziegler-Nicols Method -----
7
8 Kc = 1.32 # Critical Gain
9 w180 = 0.58 # Cross frequency for given critical gain
10 Tc = 2*math.pi/w180; # Critical Period
11
12 # Formulas for calculation PI controller
13 Kp = 0.45 * Kc
14 Ti = Tc/1.2
15
16 print ("Kp =", f'{Kp:.2f}')
17 print ("Ti =", f'{Ti:.2f}', "s")
18
19 # ----- Define Transfer Functions -----
20
21 # Transfer Function Process
22 num_p = np.array ([1])
23 den_p = np.array ([1, 1, 0])
24
25 Hp = control.tf(num_p, den_p)
26 print ('Hp(s) =', Hp)
27
28
29 # Transfer Function Sensor
30 num_m = np.array ([1])
31 den_m = np.array ([3, 1])
32
33 Hm = control.tf(num_m, den_m)
34 print ('Hm(s) =', Hm)
35
36
37 # Transfer Function PI Controller
38 num_c = np.array ([Kp*Ti, Kp])
39

```

```

40 den_c = np.array ([Ti , 0])
41
42 Hc = control.tf(num_c, den_c)
43 print ('Hc(s) =', Hc)
44
45
46
47 # The Loop Transfer function
48 L = control.series(Hc, Hp, Hm)
49 print ('L(s) =', L)
50
51 # Tracking transfer function
52 T = control.feedback(L,1)
53 print ('T(s) =', T)
54
55 # Sensitivity transfer function
56 S = 1 - T
57 print ('S(s) =', S)
58
59
60 # ----- Plotting -----
61
62 # Step Response Feedback System (Tracking System)
63 t, y = control.step_response(T)
64 plt.figure(1)
65 plt.plot(t,y)
66 plt.title("Step Response Feedback System T(s)")
67
68
69 # Bode Diagram
70 plt.figure(2)
71 control.bode(L, dB=True, deg=True)
72
73
74 # Bode Diagram with Stability Margins
75 plt.figure(3)
76 control.bode(L, dB=True, deg=True, margins=True)
77
78
79 # Poles and Zeros
80 control.pzmap(T)
81
82 p = control.pole(T)
83 z = control.zero(T)
84
85 print("poles = ", p)
86
87
88 # ----- Print Stability Analysis -----
89
90 # Calculating stability margins and crossover frequencies
91 gm , pm , w180 , wc = control.margin(L)
92
93 # Convert gm to Decibel
94 gmdb = 20 * np.log10(gm)
95
96 print("wc =", f'{wc:.2f}', "rad/s")
97 print("w180 =", f'{w180:.2f}', "rad/s")
98
99 print("GM =", f'{gm:.2f}')
100 print("GM =", f'{gmdb:.2f}', "dB")

```

```
101 print("PM =", f'{pm:.2f}', "deg")
```

Listing 33.2: PI Controller using Ziegler–Nichols Method

This gives the the plots shown in Figure 33.19.

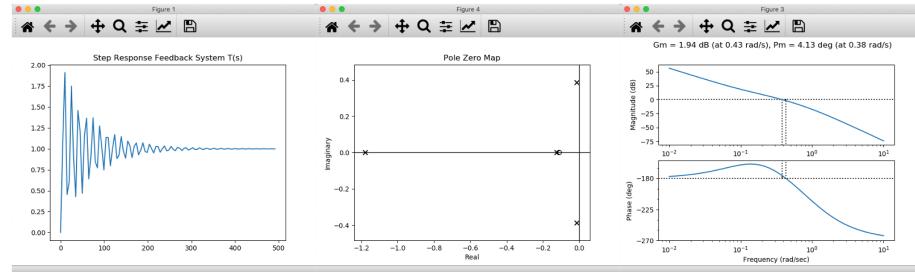


Figure 33.19: Ziegler–Nichols (PI Controller)

From the simulations we get:

$$\begin{aligned} w_c &= 0.38 \text{ rad/s} \\ w_{180} &= 0.43 \text{ rad/s} \\ GM &= 1.25 \\ GM &= 1.94 \text{ dB} \\ PM &= 4.13 \text{ deg} \end{aligned}$$

From the results we see that the system is stable but the results are not so good. The reason for this is that the process already have an integrator. Using a P control gives a much better result for this process.

#### Skogestad's method Method

The Skogestad's method (SIMC method) assumes you apply a step on the input ( $u$ ) and then observe the response and the output ( $y$ ).

If we have a model of the system, we can use the Skogestad's formulas for finding the PI(D) parameters directly.

[End of Example]

Table 33.1: Ziegler–Nichols Frequency Response method

Controller	$K_p$	$T_i$	$T_d$
P	$0.5K_c$	$\infty$	0
PI	$0.45K_c$	$\frac{T_c}{T_c^2}$	0
PID	$0.6K_c$	$\frac{T_c}{2}$	$\frac{T_c}{8}$

## Chapter 34

# Air Heater Control System

### 34.1 Introduction

The Air Heater is a small-scale laboratory process suitable for learning about control systems.

Figure 34.1 shows the Air Heater laboratory process.

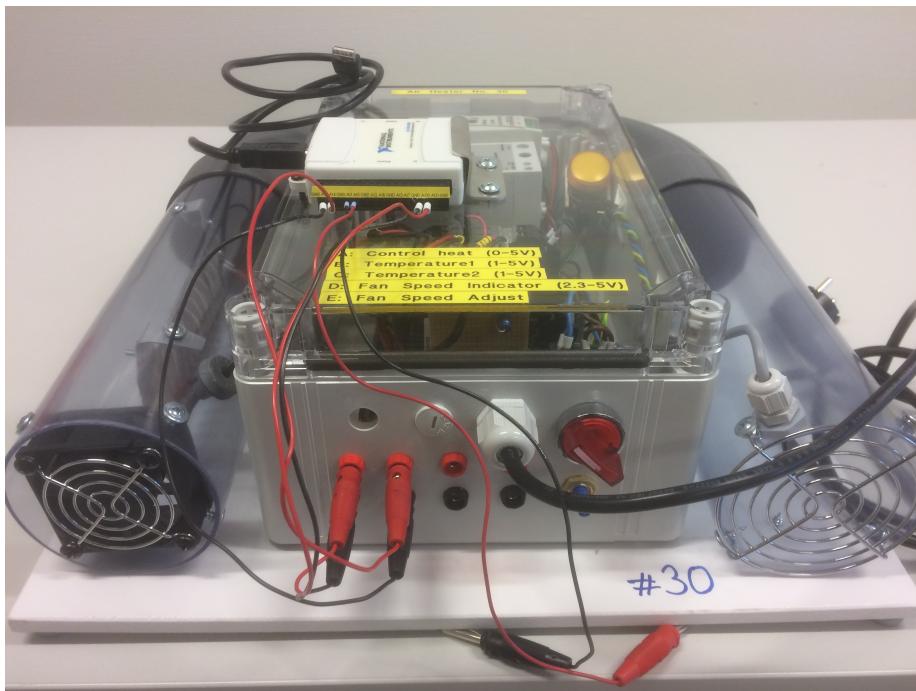


Figure 34.1: Air Heater Laboratory Process

We want to implement and control the Air Heater system in Python. We start by implementing a control system using a mathematical model of the system.

We use the mathematical model to design and test proper PID parameters. Finally, we will implement a control system using the real Air Heater which is shown in Figure 34.1.

Figure 34.2 shows a sketch of the Air Heater system and involved parameters.

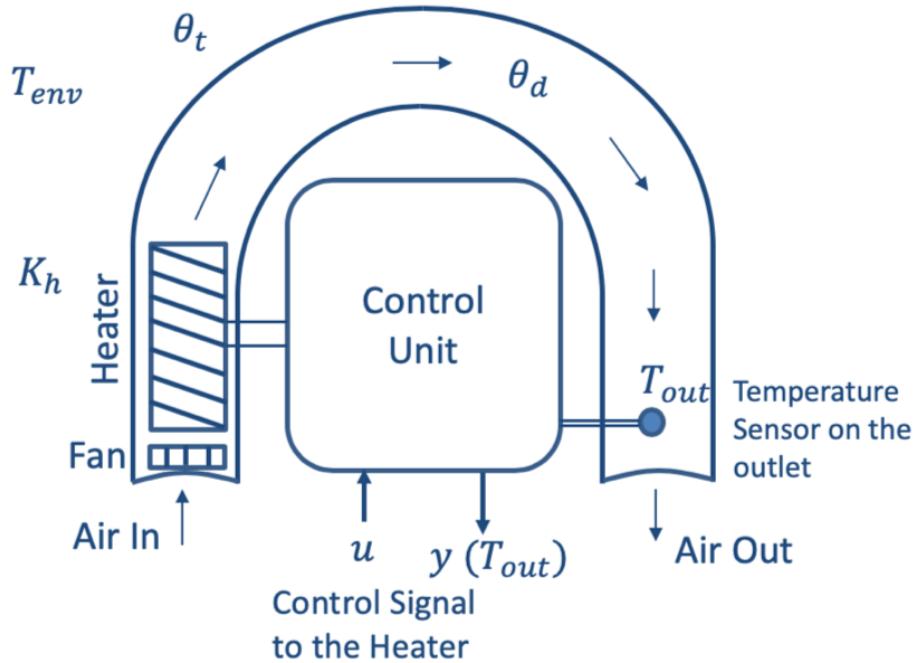


Figure 34.2: Air Heater Overview

The aim is to be able to control the temperature on the outlet ( $T_{out}$ ).

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>

The Air Heater system can be modelled as a 1. order system with time-delay (34.1):

$$\dot{T}_{out} = \frac{1}{\theta_t}(-T_{out} + K_h u(t - \theta_d) + T_{env}) \quad (34.1)$$

Where:

- $T_{out}[^{\circ}C]$  is the air temperature at the tube outlet
- $u[V]$  is the control signal to the heater
- $\theta_t[s]$  is the time-constant
- $K_h[^{\circ}C/V]$  is the heater gain
- $\theta_d[s]$  is the time-delay representing air transportation and sluggishness in the heater
- $T_{env}[^{\circ}C]$  is the environmental (room) temperature. It is the temperature in the outlet air of the air tube when the control signal to the heater has been set to zero for relatively long time (some minutes)

In our simulations we can assume the following values:

$$\theta_t = 22s \quad (34.2)$$

$$\theta_d = 2s \quad (34.3)$$

$$K_h = 3.5 \frac{^{\circ}C}{V} \quad (34.4)$$

$$T_{env} = 21.5^{\circ}C \quad (34.5)$$

## 34.2 Discrete Air Heater

We can use e.g., the Euler Approximation in order to find the discrete Model:

$$\dot{x} \approx \frac{x(k+1) - x(k)}{T_s} \quad (34.6)$$

Where:

- $T_s$  is the Sampling Time
- $x(k)$  is the present value (discrete time)
- $x(k+1)$  is next (future) value (discrete time)

The discrete Model will then be on the form:

$$x(k+1) = x(k) + \dots \quad (34.7)$$

We can then implement the discrete model in Python.

### Discretization of Air Heater:

We make a discrete version:

$$\frac{T_{out}(k+1) - T_{out}(k)}{T_s} = \frac{1}{\theta_t}(-T_{out}(k) + K_h u(k - \frac{\theta_d}{T_s}) + T_{env}) \quad (34.8)$$

Note that  $u(t)$  is  $u(k)$  in discrete version. Dealing with time delays it is a little more complicated, i.e.,  $u(t - \theta_d)$  becomes  $u(k - \frac{\theta_d}{T_s})$  in discrete version.

Assume  $\theta_d = 2s$  and  $T_s = 0.1s$ , we get  $u(k - 20)$ . This mean that we have to remember the 20 previous samples of  $u(k)$ .

This gives the following discrete system:

$$T_{out}(k+1) = T_{out}(k) + \frac{T_s}{\theta_t}(-T_{out}(k) + K_h u(k - \frac{\theta_d}{T_s}) + T_{env}) \quad (34.9)$$

Note that the time delay can be a little tricky to implement.

#### Example 34.2.1. Simulation of Air Heater without Time delay

The time delay makes the implementation a little more complicated, so we start by setting  $\theta_d = 0$

Simplified Discrete Air Heater equation ( $\theta_d = 0$ ):

$$T_{out}(k+1) = T_{out}(k) + \frac{T_s}{\theta_t}(-T_{out}(k) + K_h u(k) + T_{env}) \quad (34.10)$$

We can set the Sampling Time  $T_s = 0.1s$

Python code:

```

1 # Air Heater System
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Model Parameters
6 Kh = 3.5
7 theta_t = 22
8 theta_d = 2
9 Tenv = 21.5
10
11 # Simulation Parameters
12 Ts = 0.1 # Sampling Time
13 Tstop = 120 # End of Simulation Time
14 uk = 1 # Step Response

```

```

15 N = int(Tstop/Ts) # Simulation length
16 Tout = np.zeros(N+2) # Initialization the Tout vector
17 Tout[0] = 0 # Initial Value
18
19 # Simulation
20 for k in range(N+1):
21     Tout[k+1] = Tout[k] + (Ts/theta_t) * (-Tout[k] + Kh*uk + Tenv)
22
23
24 # Plot the Simulation Results
25 t = np.arange(0, Tstop+2*Ts, Ts) #Create the Time Series
26
27 plt.plot(t, Tout)
28
29 # Formatting the appearance of the Plot
30 plt.title('Simulation of Air Heater')
31 plt.xlabel('t [s]')
32 plt.ylabel('Tout [degC]')
33 plt.grid()
34 xmin = 0
35 xmax = Tstop
36 ymin = 0
37 ymax = 40
38 plt.axis([xmin, xmax, ymin, ymax])
39 plt.show()

```

Listing 34.1: Simulation of Air Heater

This gives the the plot shown in Figure 34.3.

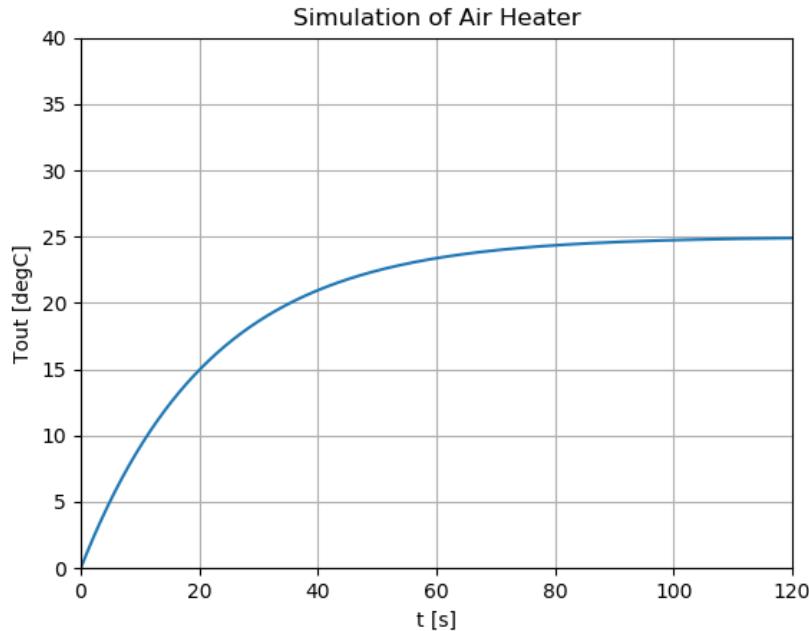


Figure 34.3: Simulation of Air Heater Model (without Time delay)

[End of Example]

**Example 34.2.2.** Simulation of Air Heater with Time delay

We will implement the full version of the Air Heater with time delay:

$$T_{out}(k+1) = T_{out}(k) + \frac{T_s}{\theta_t}(-T_{out}(k) + K_h u(k - \frac{\theta_d}{T_s}) + T_{env}) \quad (34.11)$$

Python code:

```
1 # Air Heater System
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Model Parameters
6 Kh = 3.5
7 theta_t = 22
8 theta_d = 5
9 Tenv = 21.5
10
11 # Simulation Parameters
12 Ts = 0.1 # Sampling Time
13 Tstop = 120 # End of Simulation Time
14 #uk = 1 # Step Response
15 N = int(Tstop/Ts) # Simulation length
16 Tout = np.zeros(N+2) # Initialization the Tout vector
17 Tout[0] = 20 # Initial Value
18
19 # Control Signal with Time Delay
20 N1 = int(theta_d/Ts)
21 u1 = np.zeros(N1)
22 u2 = np.ones(N)
23 uk = np.append(u1,u2)
24
25
26 # Simulation
27 for k in range(N+1):
28     Tout[k+1] = Tout[k] + (Ts/theta_t) * (-Tout[k] + Kh*uk[k] +
29     Tenv)
30
31 # Plot the Simulation Results
32 t = np.arange(0,Tstop+2*Ts,Ts) #Create the Time Series
33
34 plt.plot(t,Tout)
35
36 # Formatting the appearance of the Plot
37 plt.title('Simulation of Air Heater')
38 plt.xlabel('t [s]')
39 plt.ylabel('Tout [degC]')
40 plt.grid()
41 xmin = 0
42 xmax = Tstop
43 ymin = 20
44 ymax = 26
45 plt.axis([xmin, xmax, ymin, ymax])
```

```
46 plt.show()
```

Listing 34.2: Simulation of Air Heater with Time delay

This gives the the plot shown in Figure 34.4.

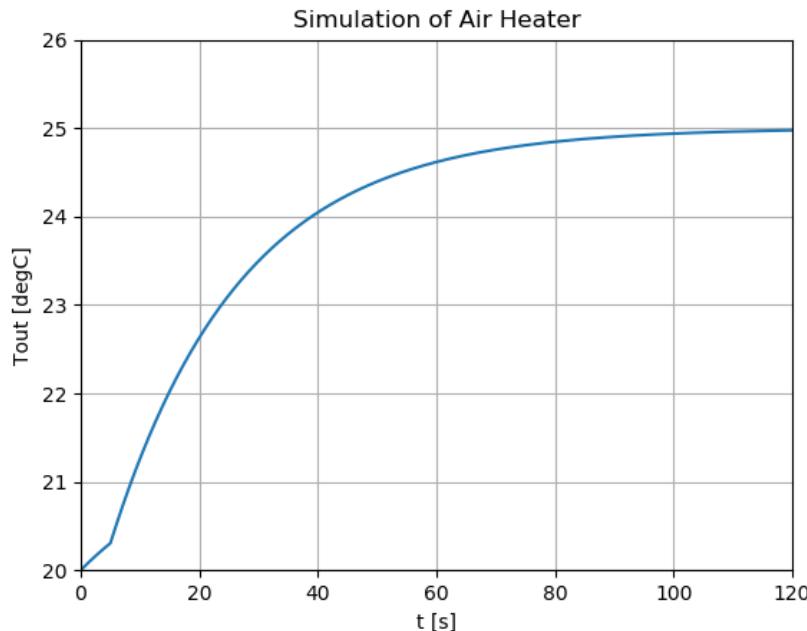


Figure 34.4: Simulation of Air Heater Model

[End of Example]

### 34.3 Transfer Function

For stability analysis and frequency response analysis we typically need to have the transfer function for the given process.

We have the differential equation as given in (34.1).

We want to find the following transfer function for the system. We use Laplace Transformation and find the transfer function from  $u(s)$  to  $T_{out}(s)$ , i.e.:

$$H(s) = \frac{T_{out}(s)}{u(s)} \quad (34.12)$$

In addition, we set all other inputs and outputs to zero, i.e., we set  $T_{env} = 0$

Some of the following Laplace transformations may be useful:

Derivation:

$$\dot{x} \iff sx(s) \quad (34.13)$$

Integration:

$$\int x \iff \frac{1}{s}x(s) \quad (34.14)$$

Time delay:

$$u(t - \tau) \iff u(s)e^{-\tau s} \quad (34.15)$$

This gives for our system (Air Heater):

$$\dot{T}_{out} = -\frac{T_{out}}{\theta_t} + \frac{K_h}{\theta_t}u(t - \theta_d) \quad (34.16)$$

We use Laplace:

$$sT_{out}(s) = -\frac{T_{out}}{\theta_t} + \frac{K_h}{\theta_t}e^{-\theta_d s}u(s) \quad (34.17)$$

Then:

$$T_{out}(s)(s + \frac{1}{\theta_t}) = \frac{K_h}{\theta_t}e^{-\theta_d s}u(s) \quad (34.18)$$

Next:

$$\frac{T_{out}(s)}{u(s)} = \frac{\frac{K_h}{\theta_t}}{s + \frac{1}{\theta_t}}e^{-\theta_d s} \quad (34.19)$$

This finally gives:

$$\frac{T_{out}(s)}{u(s)} = \frac{K}{Ts + 1} e^{-\tau s} = \frac{K_h}{\theta_t s + 1} e^{-\theta_d s} \quad (34.20)$$

Where:

Gain:

$$K = K_h \quad (34.21)$$

Time constant:

$$T = \theta_t \quad (34.22)$$

Time delay:

$$\tau =_d \quad (34.23)$$

This means the Air Heater is a standard 1.order transfer function with time delay:

$$H(s) = \frac{T_{out}(s)}{u(s)} = \frac{K_h}{\theta_t s + 1} e^{-\theta_d s} \quad (34.24)$$

Step response for a 1.order process with time delay, see Figure 34.5.

### Example 34.3.1. Simulation of Air Heater Transfer Function

```

1 # Air Heater System
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import control
5
6 # Process Parameters
7 Kh = 3.5
8 theta_t = 22
9 theta_d = 2
10
11 # Transfer Function

```

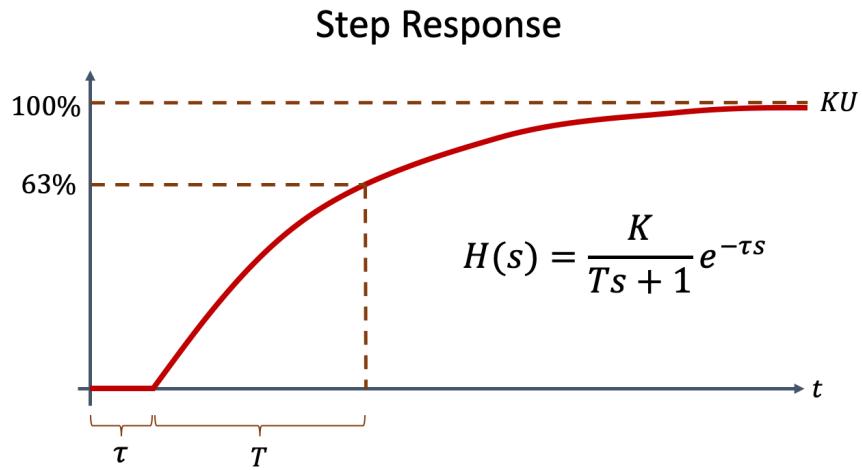


Figure 34.5: 1.order Process with Time delay

```

12 num = np.array ([Kh])
13 den = np.array ([theta_t , 1])
14 H1 = control.tf(num , den)
15 print ('H1(s) =' , H1)
16
17 N = 5 # Order of the Approximation
18 [num_pade,den_pade] = control.pade(theta_d , N)
19 Hpade = control.tf(num_pade,den_pade);
20 print ('Hpade(s) =' , Hpade)
21
22 H = control.series(H1, Hpade);
23 print ('H(s) =' , H)
24
25 # Step Response
26 t , y = control.step_response(H)
27 plt.plot(t,y)

```

Listing 34.3: Simulation of Air Heater Transfer Function

[End of Example]

This gives the the plot shown in Figure 34.6.

You should of course add titles, units, etc. in the plot.

## 34.4 Stability Analysis

The Python Control Systems Library (python-control) is a Python package that implements basic operations for analysis and design of feedback control systems.

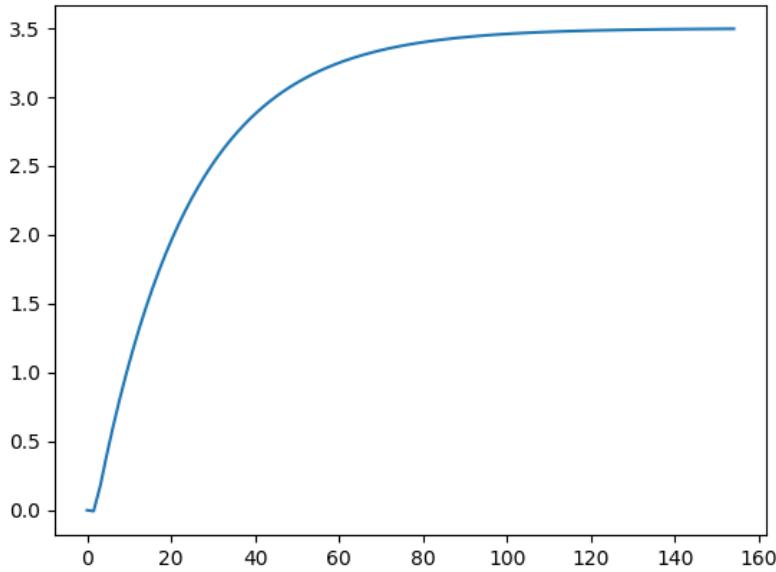


Figure 34.6: Air Heater Step Response using Transfer Function

Python Control Systems Library Documentation:

<https://python-control.readthedocs.io>

A dynamic system has one of the following stability properties:

- Asymptotically stable system
- Marginally stable system
- Unstable system

#### Example 34.4.1. Stability Analysis of Air Heater System

We will use a PI controller. The transfer function for the PI controller is:

$$H_{pi}(s) = \frac{u(s)}{e(s)} = \frac{K_p(T_i s + 1)}{T_i s} \quad (34.25)$$

We can start with the following PI parameter values:  $K_p = 0.52$  and  $T_i = 18s$  (they are found using the Skogestad method).

Below you find Python code for analysis of the stability of the Air Heater system.

Since the transfer function for the Air Heater consists of a time delay, a Pade approximation will be used.

```

1 import math
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import control
5
6 # Process Parameters
7 Kh = 3.5
8 theta_t = 22
9 theta_d = 2
10
11 # ----- Define Transfer Functions -----
12
13 # Transfer Function Process
14 num_p = np.array ([Kh])
15 den_p = np.array ([theta_t , 1])
16 Hp1 = control.tf(num_p , den_p)
17 #print ('Hp1(s) =', Hp1)
18
19 N = 5 # Time Delay - Order of the Approximation
20 [num_pade,den_pade] = control.pade(theta_d , N)
21 Hp_pade = control.tf(num_pade,den_pade);
22 #print ('Hp_pade(s) =', Hp_pade)
23
24 Hp = control.series(Hp1, Hp_pade);
25 #print ('Hp(s) =', Hp)
26
27
28 # Transfer Function PI Controller
29 Kp = 0.52
30 Ti = 18
31 num_c = np.array ([Kp*Ti, Kp])
32 den_c = np.array ([Ti , 0])
33
34 Hc = control.tf(num_c, den_c)
35 print ('Hc(s) =', Hc)
36
37
38 # The Loop Transfer function
39 L = control.series(Hc, Hp)
40 print ('L(s) =', L)
41
42 # Tracking transfer function
43 T = control.feedback(L,1)
44 print ('T(s) =', T)
45
46 # Sensitivity transfer function
47 S = 1 - T
48 print ('S(s) =', S)
49
50
51 # ----- Plotting -----
52
53 # Step Response Feedback System (Tracking System)
54 t, y = control.step_response(T)
55 plt.figure(1)
56 plt.plot(t,y)
57 plt.title("Step Response Feedback System T(s)")
58 plt.grid()

```

```

59
60
61 # Bode Diagram with Stability Margins
62 plt.figure(2)
63 control.bode(L, dB=True, deg=True, margins=True)
64
65
66 # Poles and Zeros
67 control.pzmap(T)
68
69 p = control.pole(T)
70 z = control.zero(T)
71 print("poles = ", p)
72
73 # ----- Print Stability Analysis -----
74
75 # Calculating stability margins and crossover frequencies
76 gm , pm , w180 , wc = control.margin(L)
77
78 # Convert gm to Decibel
79 gmdb = 20 * np.log10(gm)
80
81 print("wc =", f'{wc:.2f}', "rad/s")
82 print("w180 =", f'{w180:.2f}', "rad/s")
83
84 print("GM =", f'{gm:.2f}')
85 print("GM =", f'{gmdb:.2f}', "dB")
86 print("PM =", f'{pm:.2f}', "deg")
87
88
89 # Find when System is Marginally Stable (Critical Gain = Kc)
90 Kc = Kp*gm
91 print("Kc=", f'{Kc:.2f}')

```

Listing 34.4: Stability Analysis of Air Heater System

[End of Example]

Below you see the plots from the simulations.

Figure 34.7 shows the poles (and zeros) for the Air Heater feedback system.

Figure 34.8 shows the step response for the Air Heater feedback system.

Figure 34.9 shows the frequency response for the Air Heater feedback system.

Stability analysis parameters for the closed loop Air Heater System:

$w_c = 0.09 \text{ rad/s}$   
 $w_{180} = 0.78 \text{ rad/s}$   
 $GM = 9.41$   
 $GM = 19.47 \text{ dB}$   
 $PM = 75.06 \text{ deg}$   
 $K_c = 4.89$

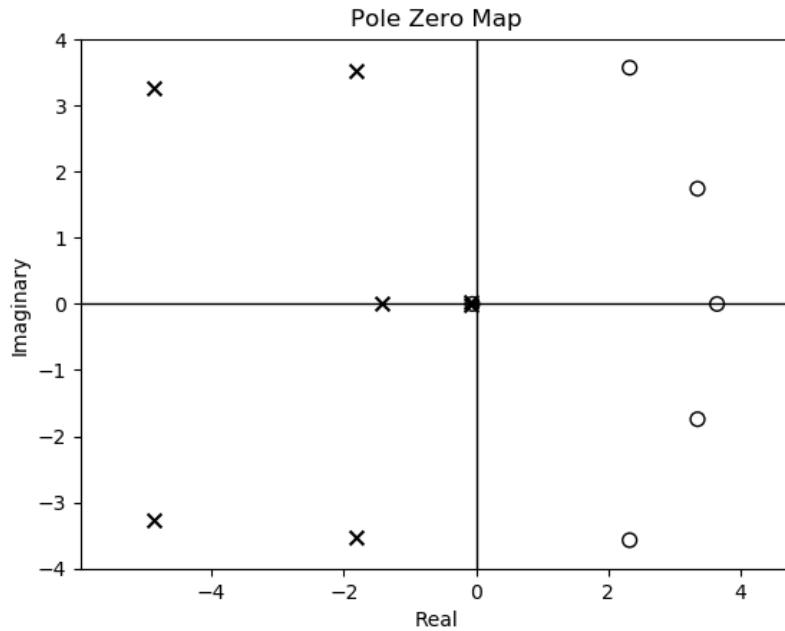


Figure 34.7: Poles and Zeros for the Air Heater System

From the results (poles, step response and frequency response) we see that the system is asymptotically stable.

”Golden rules” for the stability margins are as follows:

$$2(6dB) < \Delta K < 4(12dB) \quad (34.26)$$

$$30^\circ < \phi < 60^\circ \quad (34.27)$$

We see that our margins could be a little smaller, so it is possible to increase  $K_p$  for our control system.

How much can we increase  $K_p$  before the system gets unstable?

The critical gain is as follows:

$$K_c = K_p * \Delta K = 0.52 * 9.41 \approx 4.89 \quad (34.28)$$

This means for  $K_p = 4.89$  we will have a marginally stable system and for  $K_p$  values above that the system will become unstable.

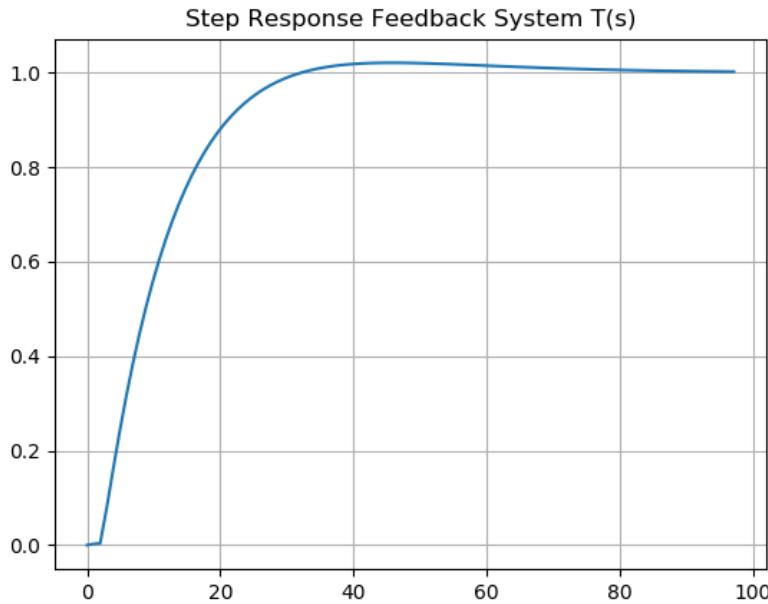


Figure 34.8: Step Response for the Air Heater System

If we change our Python program by setting  $K_p = 4.89$  and later, e.g., set  $K_p = 5$  we see that our simulations confirms that.

#### **Marginally Stable System:**

Poles: Each of the poles of the transfer function lies strictly in the left half plane (has strictly negative real part).

Step Response of Tracking Function:

$$\lim_{t \rightarrow \infty} y(t) = k \quad (34.29)$$

Frequency Response:

$$\omega_c < \omega_{189} \quad (34.30)$$

#### **Asymptotically Stable System:**

Poles: One or more poles lies on the imaginary axis (have real part equal to zero), and all these poles are distinct. Besides, no poles lie in the right half plane.

Step Response of Tracking Function:

$G_m = 19.47 \text{ dB}$  (at  $0.78 \text{ rad/s}$ ),  $P_m = 75.06 \text{ deg}$  (at  $0.09 \text{ rad/s}$ )

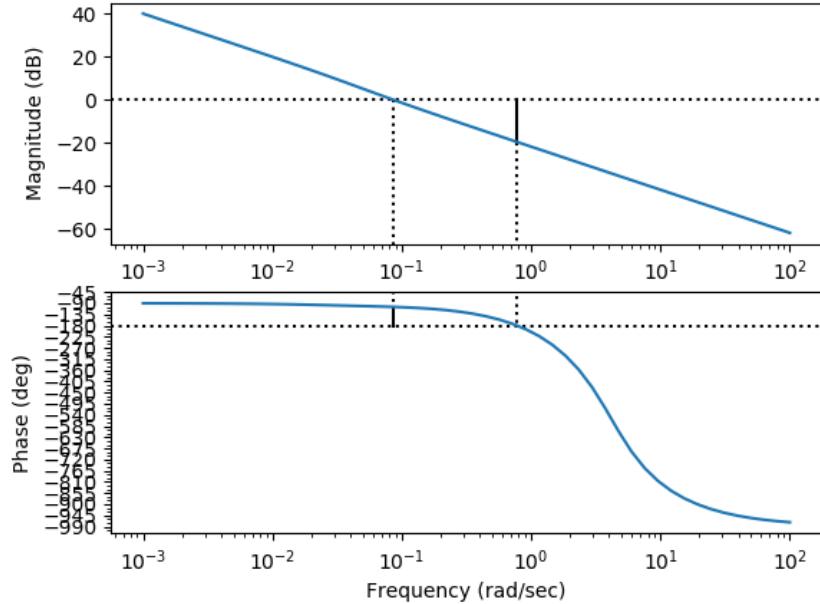


Figure 34.9: Frequency Response for the Air Heater System

$$0 < \lim_{t \rightarrow \infty} y(t) < \infty \quad (34.31)$$

Frequency Response:

$$\omega_c = \omega_{189} \quad (34.32)$$

### Unstable System:

Poles: At least one pole lies in the right half plane (has real part greater than zero). Or there are multiple and coincident poles on the imaginary axis (Example: double integrator  $H(s) = \frac{1}{s^2}$  ).

Step Response of Tracking Function:

$$\lim_{t \rightarrow \infty} y(t) = \infty \quad (34.33)$$

Frequency Response:

$$\omega_c > \omega_{189} \quad (34.34)$$

## 34.5 Ziegler–Nichols Frequency Response Method

We will use the Ziegler–Nichols Frequency Response method in order to find proper PI parameters ( $K_p$  and  $T_i$ ) for this system.

The Ziegler–Nichols Frequency Response method can shortly be described like this:

Assume you use a P controller only, i.e.,  $T_i = \infty, T_d = 0$ . Then you need to find for which  $K_p$  the closed loop system is a marginally stable system ( $\omega_c = \omega_{180}$ ). This  $K_p$  is called  $K_c$  (critical gain). The  $T_c$  (critical period) can be found from the damped oscillations of the closed loop system, as shown in Figure 34.10.

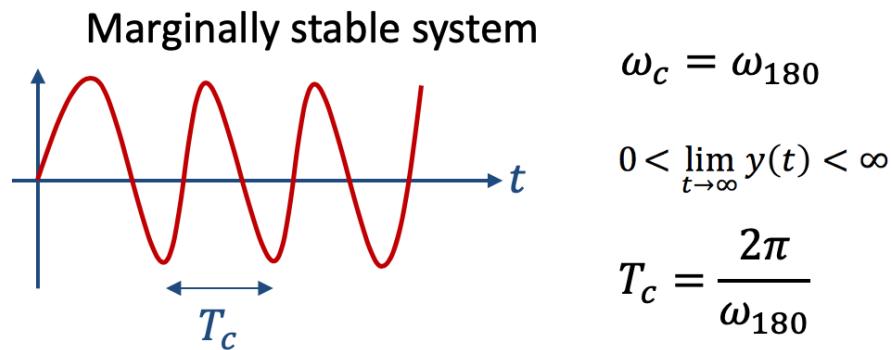


Figure 34.10: Damped Oscillations of the Closed Loop System

The critical period ( $T_c$ ) can be found (34.35):

$$T_c = \frac{2\pi}{\omega_{180}} \quad (34.35)$$

Then we can calculate the PI(D) parameters using the formulas given by Table 34.1.

Where

$K_c$  - Critical Gain  $T_c$  - Critical Period

**Example 34.5.1.** Ziegler–Nichols Frequency Response method for Air Heater System

Python code:

```

1 import math
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import control
5
6 # Process Parameters
7 Kh = 3.5
8 theta_t = 22
9 theta_d = 2
10
11 # ----- Define Transfer Functions -----
12
13 # Transfer Function Process
14 num_p = np.array ([Kh])
15 den_p = np.array ([theta_t , 1])
16 Hp1 = control.tf(num_p , den_p)
17 #print ('Hp1(s) =', Hp1)
18
19 N = 5 # Time Delay - Order of the Approximation
20 [num_pade,den_pade] = control.pade(theta_d , N)
21 Hp_pade = control.tf(num_pade,den_pade);
22 #print ('Hp_pade(s) =', Hp_pade)
23
24 Hp = control.series(Hp1, Hp_pade);
25 #print ('Hp(s) =', Hp)
26
27
28 # Transfer Function P Controller
29 Kp = 0.5
30 Kp = 5.12
31 num_c = np.array ([Kp])
32 den_c = np.array ([1])
33 Hc = control.tf(num_c, den_c)
34 #print ('Hc(s) =', Hc)
35
36
37 # The Loop Transfer function
38 L = control.series(Hc, Hp)
39 #print ('L(s) =', L)
40
41 # Tracking transfer function
42 T = control.feedback(L,1)
43 #print ('T(s) =', T)
44
45 # Sensitivity transfer function
46 S = 1 - T
47 #print ('S(s) =', S)
48
49
50 # ----- Plotting -----
51
52 # Step Response Feedback System (Tracking System)
53 t , y = control.step_response(T)
54 plt.figure(1)
55 plt.plot(t,y)
56 plt.title("Step Response Feedback System T(s)")
57 plt.grid()
58
59
60
61 # Bode Diagram with Stability Margins
62 plt.figure(2)

```

```

63 control.bode(L, dB=True, deg=True, margins=True)
64
65
66 # Poles and Zeros
67 control.pzmap(T)
68
69 p = control.pole(T)
70 z = control.zero(T)
71
72 #print(" poles = ", p)
73
74
75 # ----- Print Stability Analysis -----
76
77 # Calculating stability margins and crossover frequencies
78 gm , pm , w180 , wc = control.margin(L)
79
80 # Convert gm to Decibel
81 gmdb = 20 * np.log10(gm)
82
83 print("wc =", f'{wc:.2f}', "rad/s")
84 print("w180 =", f'{w180:.2f}', "rad/s")
85
86 print("GM =", f'{gm:.2f}')
87 print("GM =", f'{gmdb:.2f}', "dB")
88 print("PM =", f'{pm:.2f}', "deg")
89
90
91 # ----- Ziegler-Nichols Method PI Controller -----
92 # Find when System is Marginally Stable (Critical Gain Kc)
93 Kc = Kp*gm
94 print("Kc=", f'{Kc:.2f}')
95
96 Tc = (2*math.pi)/w180
97 print("Tc=", f'{Tc:.2f}')

```

Listing 34.5: Ziegler–Nichols Frequency Response method

From simulations we find that:

$$K_c \approx 5.12 \quad (34.36)$$

From the formula we find that:

$$T_c = \frac{2\pi}{\omega_{180}} \approx 7.73 \quad (34.37)$$

Ziegler–Nichols PI Controller becomes:

$$K_p = 0.45K_c = 0.45 * 5.12 \approx 2.3 \quad (34.38)$$

$$T_i = \frac{T_c}{1.2} = \frac{7.73}{1.2} \approx 6.44s \quad (34.39)$$

[End of Example]

## 34.6 Air Heater Control System

PI controller:

$$u(t) = K_p e + \frac{K_p}{T_i} \int_0^t e d\tau \quad (34.40)$$

Where  $e = r - y$ .  $r$  is the reference value or the set-point and  $y$  is the current measurement of the process value.

The transfer function for the PI controller is:

$$H_{pi}(s) = \frac{u(s)}{e(s)} = \frac{K_p(T_i s + 1)}{T_i s} \quad (34.41)$$

Discrete PI controller:

Based on the continuous PI controller above, lets create a discrete version:

We derive both sides in order to remove the integral:

$$\dot{u} = K_p \dot{e} + \frac{K_p}{T_i} e \quad (34.42)$$

Now we can use the Euler Backward method:

$$\dot{x} = \frac{x_k - x_{k-1}}{T_s} \quad (34.43)$$

Where  $T_s$  is the sampling time.

This gives:

$$\frac{u_k - u_{k-1}}{T_s} = K_p \frac{e_k - e_{k-1}}{T_s} + \frac{K_p}{T_i} e_k \quad (34.44)$$

Finally, the discrete PI controller becomes:

$$u_k = u_{k-1} + K_p(e_k - e_{k-1}) + \frac{K_p}{T_i}e_k \quad (34.45)$$

Where  $e_k = r_k - y_k$

#### Example 34.6.1. Air Heater Control System without Time delay

Simplified Discrete Air Heater equation ( $\theta_d = 0$ ):

$$T_{out}(k+1) = T_{out}(k) + \frac{T_s}{\theta_t}(-T_{out}(k) + K_h u(k) + T_{env}) \quad (34.46)$$

Python code:

```

1 # Air Heater System
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Model Parameters
6 Kh = 3.5
7 theta_t = 22
8 theta_d = 2
9 Tenv = 21.5
10
11 # Simulation Parameters
12 Ts = 0.1 # Sampling Time
13 Tstop = 120 # End of Simulation Time
14 #uk = 1 # Step Response
15 N = int(Tstop/Ts) # Simulation length
16 Tout = np.zeros(N+2) # Initialization the Tout vector
17 Tout[0] = 20 # Initial Value
18
19 # PI Controller Settings
20 # PI Parameters found from Skogestad
21 Kp = 0.51
22 Ti = 18
23 # PI Parameters found from Ziegler Nichols
24 #Kp = 2.3
25 #Ti = 6.44
26
27 r = 28 # Reference value [degC]
28 e = np.zeros(N+2) # Initialization
29 u = np.zeros(N+2) # Initialization
30
31
32 # Simulation
33 for k in range(N+1):
34     e[k] = r - Tout[k]
35     u[k] = u[k-1] + Kp*(e[k] - e[k-1]) + (Kp/Ti)*e[k]
36     if u[k]>5:
37         u[k] = 5
38     Tout[k+1] = Tout[k] + (Ts/theta_t) * (-Tout[k] + Kh*u[k] + Tenv
39 )

```

```

40
41 # Plot the Simulation Results
42 t = np.arange(0,Tstop+2*Ts,Ts) #Create the Time Series
43
44 plt.figure(1)
45 plt.plot(t,Tout)
46
47 # Plot Process Value
48
49 # Formatting the appearance of the Plot
50 plt.title('Simulation of Air Heater')
51 plt.xlabel('t [s]')
52 plt.ylabel('Tout [degC]')
53 plt.grid()
54 xmin = 0
55 xmax = Tstop
56 ymin = 20
57 ymax = 32
58 plt.axis([xmin, xmax, ymin, ymax])
59 plt.show()
60
61
62 # Plot Control Signal
63 plt.figure(2)
64 plt.plot(t,u)
65
66 # Formatting the appearance of the Plot
67 plt.title('Control Signal')
68 plt.xlabel('t [s]')
69 plt.ylabel('u [V]')
70 plt.grid()
71 xmin = 0
72 xmax = Tstop
73 ymin = 0
74 ymax = 10
75 #plt.axis([xmin, xmax, ymin, ymax])

```

Listing 34.6: Air Heater Control System without Time delay

[End of Example]

### Example 34.6.2. Air Heater Control System with Time delay

We will implement the full version of the Air Heater with time delay:

$$T_{out}(k+1) = T_{out}(k) + \frac{T_s}{\theta_t}(-T_{out}(k) + K_h u(k - \frac{\theta_d}{T_s}) + T_{env}) \quad (34.47)$$

Python code:

```

1 # Air Heater System
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Model Parameters

```

```

6 Kh = 3.5
7 theta_t = 22
8 theta_d = 2
9 Tenv = 21.5
10
11 # Simulation Parameters
12 Ts = 0.1 # Sampling Time
13 Tstop = 200 # End of Simulation Time
14 #uk = 1 # Step Response
15 N = int(Tstop/Ts) # Simulation length
16 Tout = np.zeros(N+2) # Initialization the Tout vector
17 Tout[0] = 20 # Initial Value
18
19 # PI Controller Settings
20 Kp = 0.1
21 Ti = 22
22
23 r = 28 # Reference value [degC]
24 e = np.zeros(N+2) # Initialization
25 u = np.zeros(N+2) # Initialization
26
27
28 # Simulation
29 for k in range(N+1):
30     e[k] = r - Tout[k]
31     u[k] = u[k-1] + Kp*(e[k] - e[k-1]) + (Kp/Ti)*e[k]
32     if u[k]>5:
33         u[k] = 5
34     Tout[k+1] = Tout[k] + (Ts/theta_t) * (-Tout[k] + Kh*u[int(k-
35     theta_d/Ts)] + Tenv)
36
37 # Plot the Simulation Results
38 t = np.arange(0,Tstop+2*Ts,Ts) #Create the Time Series
39
40 plt.figure(1)
41 plt.plot(t,Tout)
42
43 # Plot Process Value
44
45 # Formatting the appearance of the Plot
46 plt.title('Simulation of Air Heater')
47 plt.xlabel('t [s]')
48 plt.ylabel('Tout [degC]')
49 plt.grid()
50 xmin = 0
51 xmax = Tstop
52 ymin = 20
53 ymax = 32
54 plt.axis([xmin, xmax, ymin, ymax])
55 plt.show()
56
57
58 # Plot Control Signal
59 plt.figure(2)
60 plt.plot(t,u)
61
62 # Formatting the appearance of the Plot
63 plt.title('Control Signal')
64 plt.xlabel('t [s]')
65 plt.ylabel('u [V]')
66 plt.grid()

```

```
67 xmin = 0
68 xmax = Tstop
69 ymin = 0
70 ymax = 10
71 #plt.axis([xmin, xmax, ymin, ymax])
```

Listing 34.7: Air Heater Control System with Time delay

[End of Example]

Table 34.1: Ziegler–Nichols Frequency Response method

Controller	$K_p$	$T_i$	$T_d$
P	$0.5K_c$	$\infty$	0
PI	$0.45K_c$	$\frac{T_c}{T_c^2}$	0
PID	$0.6K_c$	$\frac{T_c}{2}$	$\frac{T_c}{8}$

# **Part IX**

# **Python Database Development**

## Chapter 35

# 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 35.1 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/>

### 35.1 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

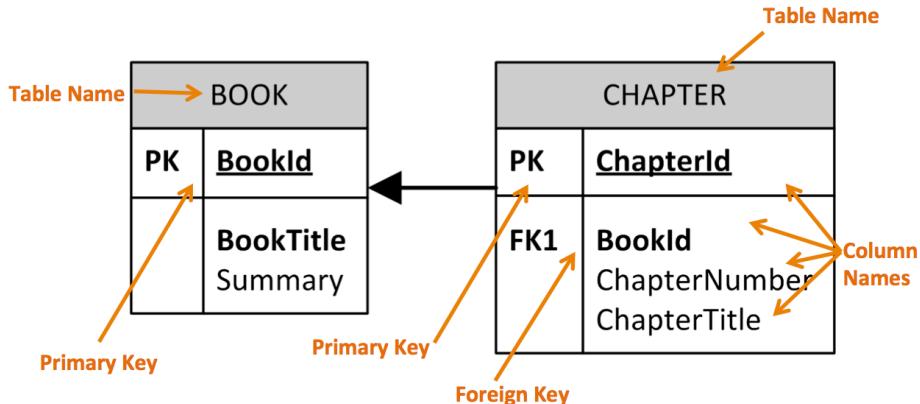


Figure 35.1: ER Diagram Example

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/>

## 35.2 SQL Server

Here we will see how we can communicate with a SQL Server database from Python.

## 35.3 MySQL

Here we will see how we can communicate with a MySQL database from Python.

## 35.4 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 36

## SQL Server with Python

### 36.1 Introduction to SQL Server

Here we will see how we can communicate with a SQL Server from Microsoft using the Python programming language.

### 36.2 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.

### 36.3 pyodbc

pyODBC uses the Microsoft ODBC driver for SQL Server.

#### 36.3.1 Installation of pyodbc

Install pyodbc using pip - Python package manager.

```
pip install pyodbc
```

#### 36.3.2 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
```

## 36.4 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 36.4.1. 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 = "NUCHPHV\SQLEXPRESS"
4 database = "BOOKS"
5 username = "sa"
6 password = "xxx"
7 conn = pyodbc.connect("DRIVER={ODBC Driver 17 for SQL Server};"
8     SERVER=" + server + ";DATABASE=" + database + ";UID=" +
9     username + ";PWD=" + password )
10
11 cursor = conn.cursor()
12
13 cursor.execute("SELECT @@version;")
14 row = cursor.fetchone()
15 while row:
16     print(row[0])
17     row = cursor.fetchone()
```

Listing 36.1: Basic SQL Server Example

### Example 36.4.2. 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 = "NUCHPHV\SQLEXPRESS"
4 database = "BOOKS"
5 username = "sa"
6 password = "xxx"
7 conn = pyodbc.connect("DRIVER={ODBC Driver 17 for SQL Server};"
8     SERVER=" + server + ";DATABASE=" + database + ";UID=" +
9     username + ";PWD=" + password )
```

```
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 36.2: Getting Data from SQL Server

[End of Example]

## 36.5 Stored Procedures

## 36.6 Resources

<https://github.com/mkleehammer/pyodbc/wiki/Connecting-to-SQL-Server-from-Windows>

## 36.7 pymssql

Resources:

<https://pypi.org/project/pymssql/>  
<http://www.pymssql.org/>

## 36.8 Resources

<https://docs.microsoft.com/en-us/sql/connect/python/python-driver-for-sql-server>

**Part X**

**Python Application  
Development**

## Chapter 37

# OPC Communication with Python

### 37.1 Introduction to OPC

OPC is a standard that defines the communication of data between devices from different manufacturers. OPC requires an OPC server that communicates with the OPC clients.

Figure 37.1 shows the basic concepts of OPC.

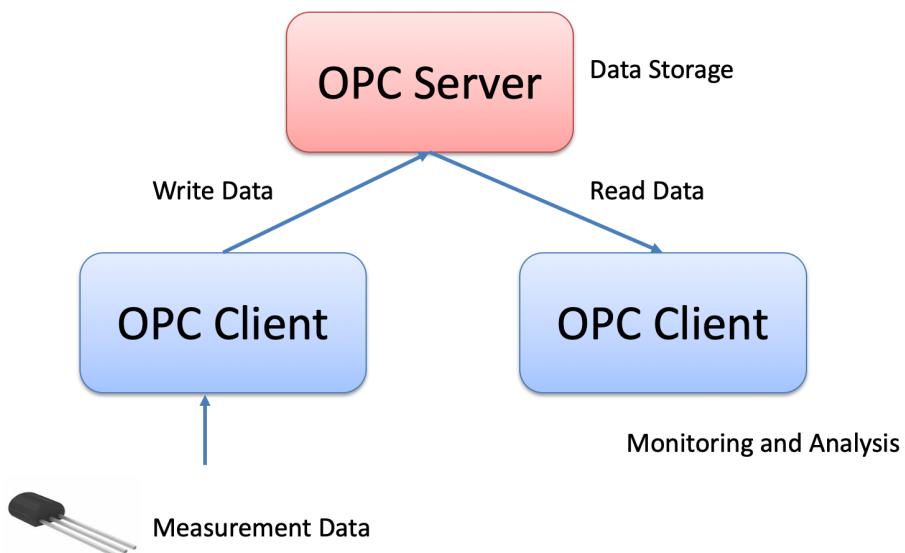


Figure 37.1: OPC

OPC allows “plug-and-play”, gives benefits as reduces installation time and the opportunity to choose products from different manufacturers.

We have different OPC standards: "Real-time" data (OPC DA), Historical data (OPC HDA), Alarm Event data (OPC AE), etc.

In Figure 37.2 we see a typical OPC scenario.

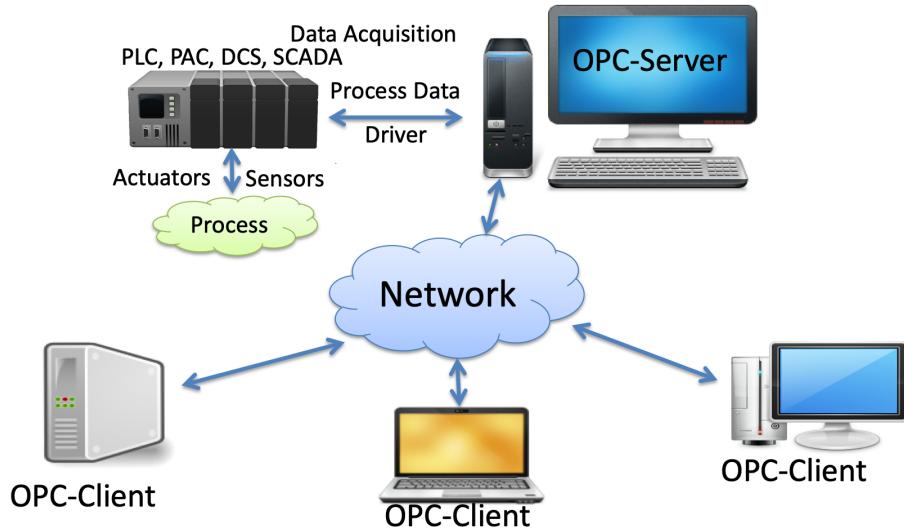


Figure 37.2: Typical OPC Scenario

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 37.3 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/>

## 37.2 OPC Classic

The most used "classic" OPC standards are the following:

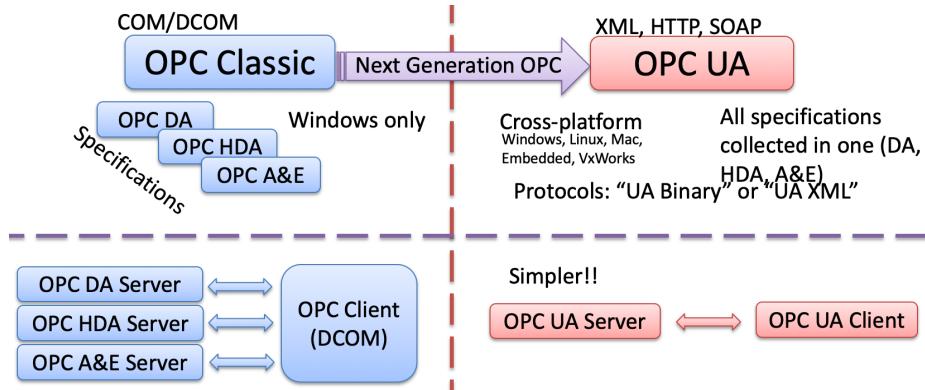


Figure 37.3: Typical OPC Scenario

- 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)

### 37.3 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

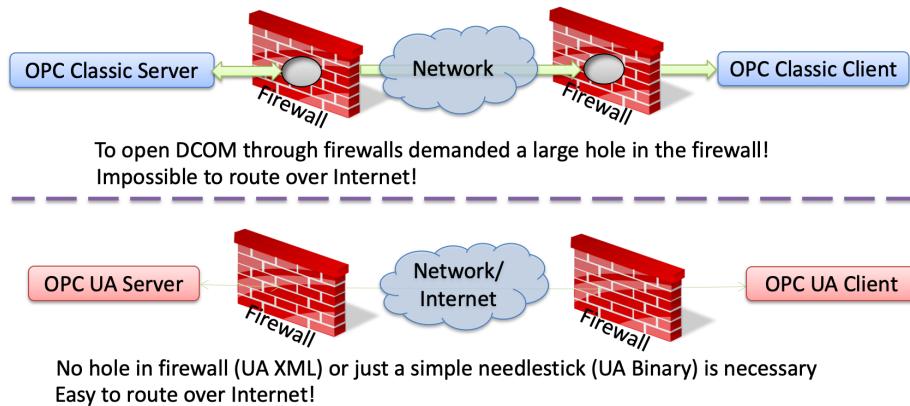


Figure 37.4: OPC Firewall Challenges

In Figure 37.4 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://Server`This uses a simple binary protocol
- “UA XML” protocol `http://Server`This 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.

## 37.4 OPC Examples with Python

## Chapter 38

# Python Integration with LabVIEW

### 38.1 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/>

Here you find information about LabVIEW, you find lots of resources like training material, videos, code examples, etc.

### 38.2 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:

[http://zone.ni.com/reference/en-XX/help/371361R-01/glang/python\\_pal/](http://zone.ni.com/reference/en-XX/help/371361R-01/glang/python_pal/)

Use the Python functions to call Python code from LabVIEW. See Figure 38.1

**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

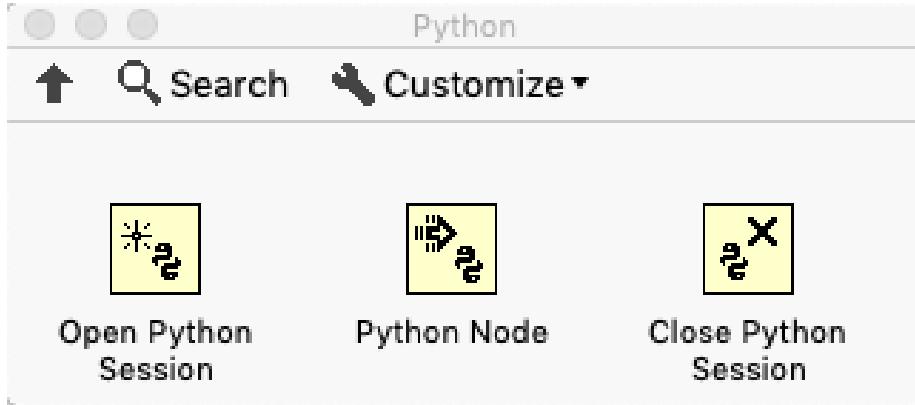


Figure 38.1: Python Integration in LabVIEW

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 38.2.1. Python Integration in LabVIEW

We want to use Python to convert between Celsius and Fahrenheit (and vice versa).

The formula for converting from Celsius to Fahrenheit is:

$$T_f = (T_c \times 9/5) + 32 \quad (38.1)$$

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \quad (38.2)$$

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 38.1: 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 38.2: 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 38.2 we see the Front Panel.

In Figure 38.3 we see the Block Diagram.

In Figure 38.4 we see LabVIEW Code for calling both Python functions (c2f and f2c) from LabVIEW.

[End of Example]

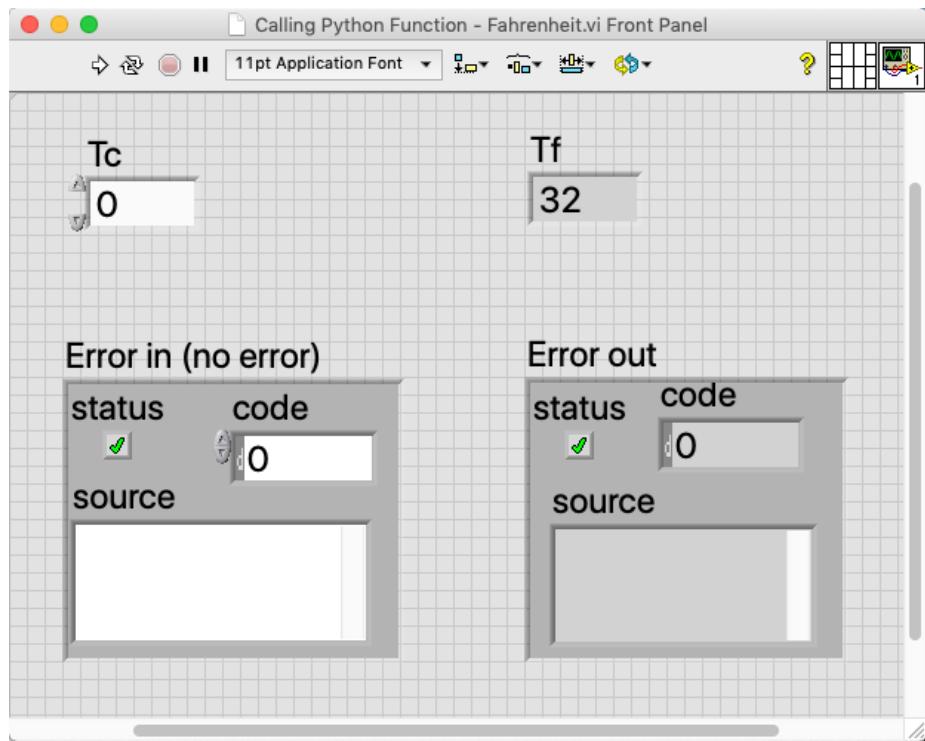


Figure 38.2: Python Integration in LabVIEW

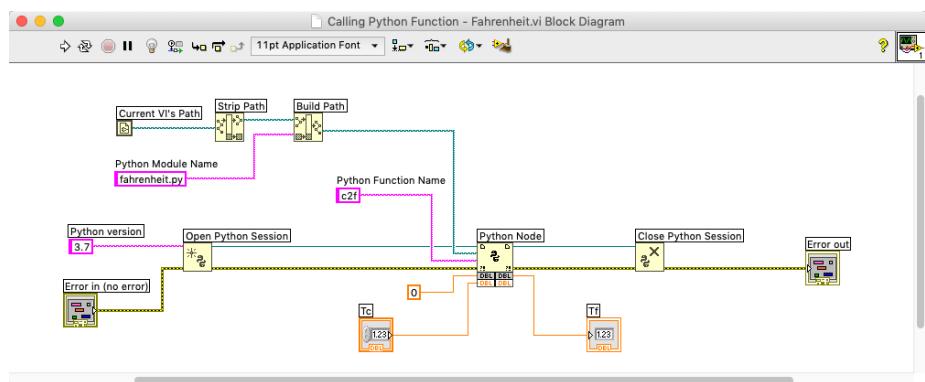


Figure 38.3: Python Integration in LabVIEW

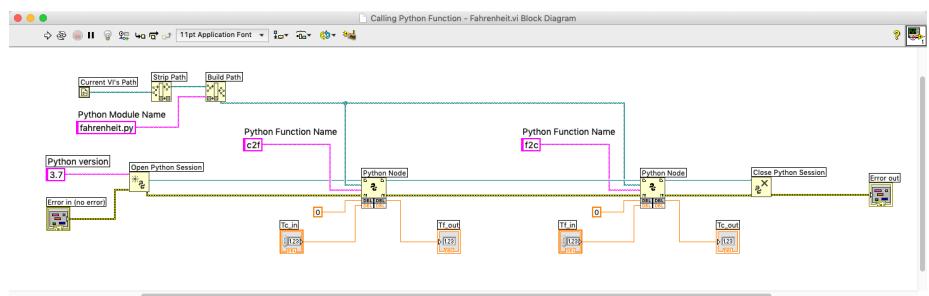


Figure 38.4: Python Integration in LabVIEW

## Chapter 39

# Raspberry Pi and Python

### 39.1 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](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 40

# 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 40.1 shows the book covers.

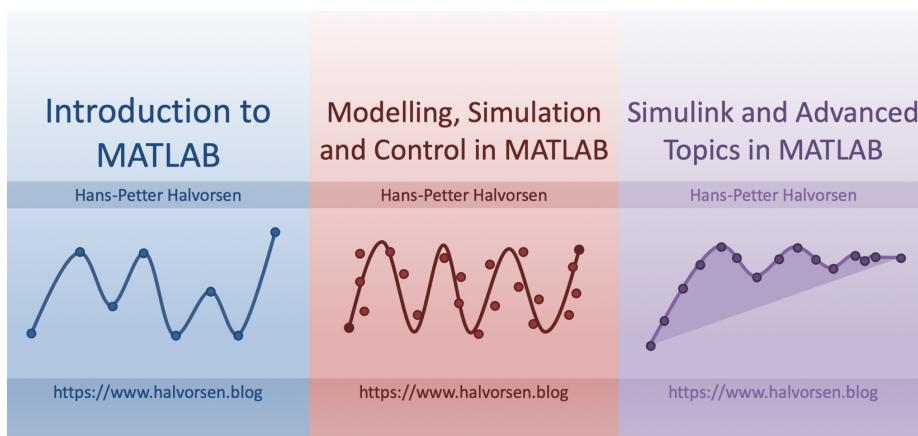


Figure 40.1: MATLAB Textbooks

## 40.1 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\
5                  Python37_64\pythonw.exe'
6     library: 'C:\Users\hansha\AppData\Local\Programs\Python\
7                  Python37_64\python37.dll'
8     home: 'C:\Users\hansha\AppData\Local\Programs\Python\
9                  Python37_64'
10    isloaded: 1
```

If MATLAB and Python don't have 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 40.1.1. 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:

$$T_f = (T_c \times 9/5) + 32 \quad (40.1)$$

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \quad (40.2)$$

First, we created 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 40.1: 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 40.2: 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 40.3: Calling Python Library and Functions from MATLAB

[End of Example]

## 40.2 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).

For more information, see the following:

<https://se.mathworks.com/help/matlab/matlab-engine-for-python.html>

# **Chapter 41**

# **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>

## **41.1 Python Distributions**

Anaconda:  
<https://www.anaconda.com>

## **41.2 Python Libraries**

NumPy Library:  
<http://www.numpy.org>

SciPy Library:  
<https://www.scipy.org>

Matplotlib Library:  
<https://matplotlib.org>

## **41.3 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>

## 41.4 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\\_Beginner](https://en.wikibooks.org/wiki/A_Beginner)

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/>

## 41.5 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\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)),” 2018.
- [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:

$$\dot{x} = ax \quad (1)$$

Where:

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

where T is the time constant.

The solution for the differential equation is:

$$x(t) = e^{at}x_0 \quad (2)$$

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:

```
1 import math as mt
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5
6 # Model Parameters
7 T = 5
8 a = -1/T
9
10 # Simulation Parameters
11 x0 = 1
12 t = 0
13
14 tstart = 0
```

```

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.

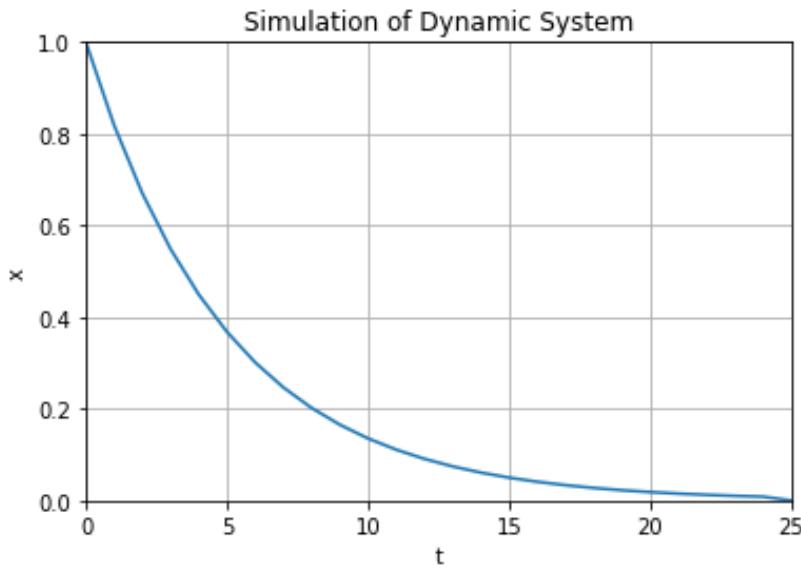


Figure 1: Simulation of Dynamic System

[End of Exercise]

# Mathematics in Python

## Create Mathematical Expressions in Python

Create a function that calculates the following mathematical expression:

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

Test with different values for x and y.

We create a Python Module with a Python Function (mymathfunctions.py):

```
1 import math as mt
2
3 def calceexpression(x,y):
4
5     z = 3*x**2 + mt.sqrt(x**2 + y**2) + mt.exp(mt.log(x))
6     return z
```

Then we can create a Python Script in order to test the function:

```
1 import mymathfunctions as mymath
2
3 x = 2
4 y = 2
5
6 z = mymath.calceexpression(x,y)
7
8 print(z)
```

The results become:

```
1 16.82842712474619
```

[End of Solution]

## Create advanced Mathematical Expressions in Python

Create the following expression in Python:

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

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 = ...  
den = ...  
f = ...
```

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:

$$\text{birth rate} = bx \quad (5)$$

$$\text{death rate} = px^2 \quad (6)$$

Then the total rate of change of bacteria population is:

$$\dot{x} = bx - px^2 \quad (7)$$

Set  $b=1/\text{hour}$  and  $p=0.5 \text{ bacteria-hour}$

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:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s} \quad (8)$$

$T_s$  - Sampling Interval

Then we get:

$$\frac{x_{k+1} - x_k}{T_s} = bx_k - px_k^2 \quad (9)$$

This gives the following discrete differential equation:

$$x_{k+1} = x_k + T_s(bx_k - px_k^2) \quad (10)$$

Now we are ready to simulate the system.

#### Python Script:

```

1 # Simulation of Bacteria Population
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 = 0.01
11 Tstop = 1
12 xk = 100
13 N = int(Tstop/Ts) # Simulation length
14 data = []
15 data.append(xk)
16
17 # Simulation
18 for k in range(N):
19     xk1 = xk + Ts* (b * xk - p * xk**2);
20     xk = xk1
21     data.append(xk1)
22
23 # Plot the Simulation Results
24 t = np.arange(0, Tstop+Ts, Ts)
25
26 plt.plot(t, data)
27 plt.title('Simulation of Bacteria Population')
28 plt.xlabel('t [s]')
29 plt.ylabel('x')
30 plt.grid()
31 plt.axis([0, 1, 0, 100])
32 plt.show()
```

The simulation gives the results as shown in Figure 2.

[End of Solution]

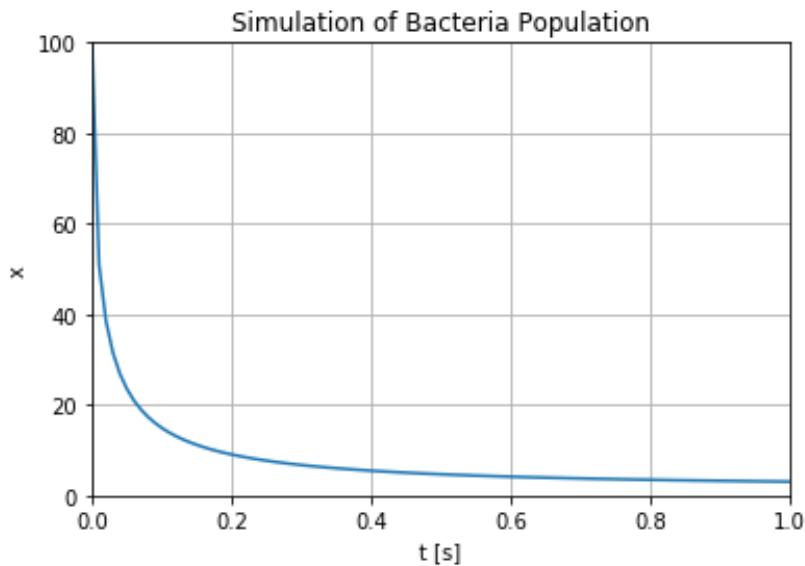


Figure 2: Simulation of Bacteria Population

## Simulation with 2 variables

Given the following system:

$$\frac{dx_1}{dt} = -x_2 \quad (11)$$

$$\frac{dx_2}{dt} = x_1 \quad (12)$$

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:

```

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 x1k = 1
14 x2k = 1
15 N = int((Tstop-Tstart)/Ts) # Simulation length
16 datax1 = []

```

```

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.

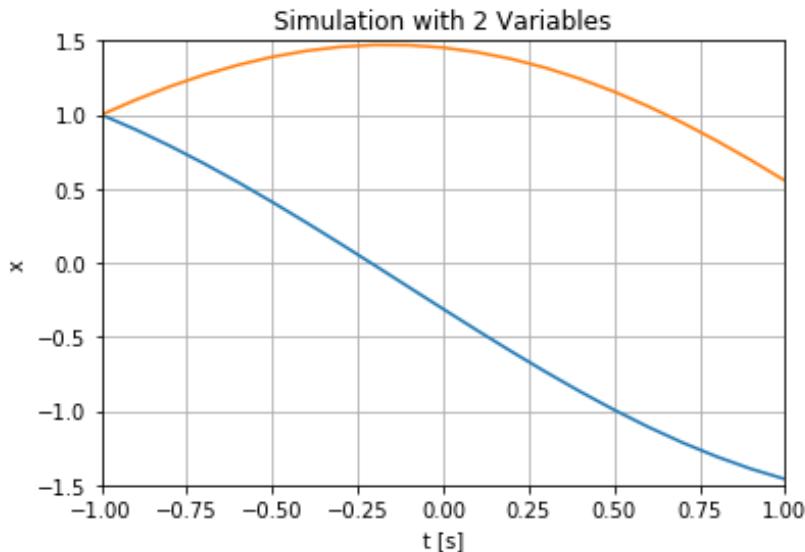


Figure 3: Simulation Example with 2 Variables

#### 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