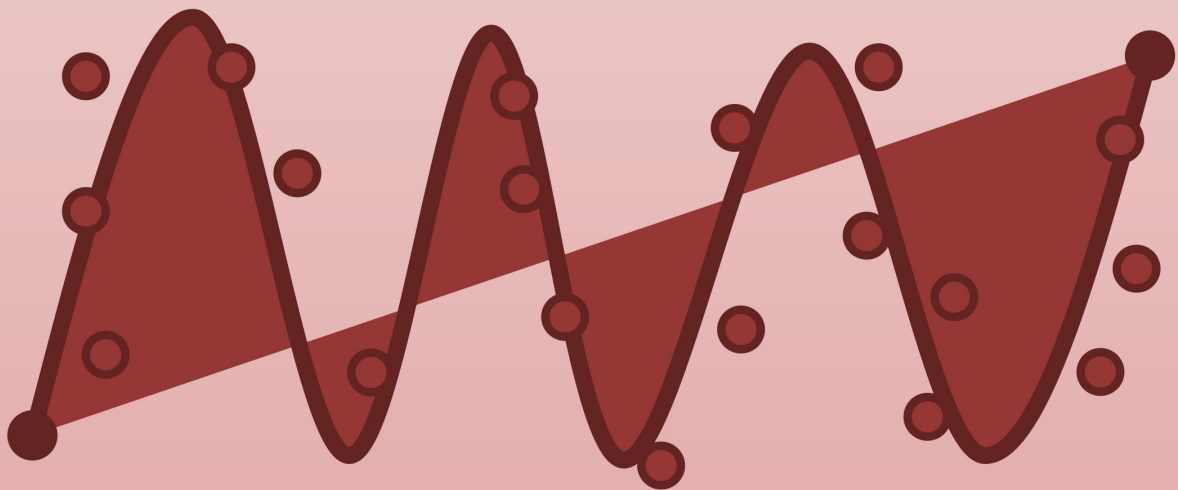


Python for Science and Engineering

Hans-Petter Halvorsen



<https://www.halvorsen.blog>



www.vistacompany.ir

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 as Windows, macOS, and Linux.

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

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

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



Here you find my Web page with Python resources:

<https://www.halvorsen.blog/documents/programming/python/>

These resources are a supplement to this textbook. Here you can download the software, download code examples, etc.

This Textbook is written in \LaTeX using Overleaf.

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

For more information about L^AT_EX:
<https://www.latex-project.org>

Overleaf is a web-bases L^AT_EXsystem, meaning you can write your L^AT_EXdocuments 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/software_engineering/

Contents

I	Getting Started with Python	13
1	Introduction	14
1.1	The New Age of Programming	14
1.2	MATLAB	18
2	What is Python?	20
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
3	Start using Python	29
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 Win- dows	36

3.5.4	Run Python Scripts from Spyder	36
4	Basic Python Programming	39
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
II	Python Programming	53
5	Python Programming	54
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
6	Creating Functions in Python	63
6.1	Introduction	63
6.2	Functions with multiple return values	65
6.3	Exercises	66
7	Creating Classes in Python	69
7.1	Introduction	69
7.2	The <code>__init__()</code> Function	70
7.3	Exercises	73
8	Creating Python Modules	74
8.1	Python Modules	74
8.2	Exercises	75
9	File Handling in Python	77
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

10 Error Handling in Python	82
10.1 Introduction to Error Handling	82
10.1.1 Syntax Errors	82
10.1.2 Exceptions	82
10.2 Exceptions Handling	83
11 Debugging in Python	85
12 Installing and using Python Packages	86
12.1 What is PIP?	86
III Python Environments and Distributions	87
13 Introduction to Python Environments and Distributions	88
13.1 Package and Environment Managers	89
13.1.1 PIP	89
13.1.2 Conda	89
13.2 Python Virtual Environments	90
14 Anaconda	91
14.1 Anaconda Navigator	91
15 Enthought Canopy	93
IV Python Editors	94
16 Python Editors	95
17 Spyder	97
18 Visual Studio Code	99
18.1 Introduction to Visual Studio Code	99
18.2 Python in Visual Studio Code	100
19 Visual Studio	101
19.1 Introduction to Visual Studio	101
19.2 Work with Python in Visual Studio	101
19.2.1 Make Visual Studio ready for Python Programming . . .	102
19.2.2 Python Interactive	102
19.2.3 New Python Project	103
20 PyCharm	109
21 Wing Python IDE	111
22 Jupyter Notebook	113
22.1 JupyterHub	114
22.2 Microsoft Azure Notebooks	114

V	Python for Mathematics Applications	116
23	Mathematics in Python	117
23.1	Basic Math Functions	117
23.1.1	Exercises	119
23.2	Statistics	121
23.2.1	Introduction to Statistics	121
23.2.2	Statistics functions in Python	122
23.3	Trigonometric Functions	124
23.4	Polynomials	128
24	Linear Algebra in Python	131
24.1	Introduction to Linear Algebra	131
24.2	Linear Algebra with Python	132
24.2.1	Vectors	133
24.2.2	Matrices	134
24.2.3	Linear Algebra (numpy.linalg)	134
24.2.4	Matrix Addition	134
24.2.5	Matrix Subtraction	135
24.2.6	Matrix Multiplication	136
24.2.7	Transpose of a Matrix	139
24.2.8	Determinant	140
24.2.9	Inverse Matrix	141
24.3	Solving Linear Equations	142
24.4	Exercises	144
25	Complex Numbers in Python	146
25.1	Introduction to Complex Numbers	146
25.2	Complex Numbers with Python	148
26	Differential Equations	150
26.1	Introduction to Differential Equations	150
26.2	ODE Solvers in Python	153
26.3	Solving Multiple 1. order Differential Equations	156
26.4	Solving Higher order Differential Equations	159
26.5	Exercises	161
27	Interpolation	166
27.1	Exercises	169
28	Curve Fitting - Fitting Models to Data	172
28.1	Linear Regression	172
28.2	Polynomial Regression	174
28.3	Exercises	180
29	Least Square Method	183
30	Numerical Differentiation	187
30.1	Differentiation on Polynomials	193

31 Numerical Integration	197
31.1 Integration on Polynomials	202
31.2 Exercises	203
32 Optimization	205
 VI Using Python for Simulations	 209
33 Introduction to Simulations	210
34 Differential Equations	211
34.1 Introduction to Differential Equations	211
35 Discrete Systems	213
35.1 Discretization	213
35.2 Exercises	217
36 Real-Time Simulations	219
36.1 Introduction	219
36.2 Introduction to Real-Time Plotting	221
36.3 Real-Time Plotting with Animation	226
36.3.1 Speeding Up the Plot Animation	230
 VII Data Acquisition (DAQ) with Python	 235
37 Plotting Sensor Data	236
37.1 Introduction	236
37.2 Introduction to Real-Time Plotting	236
37.3 Real-Time Plotting with Animation	238
37.3.1 Speeding Up the Plot Animation	240
38 Data Acquisition (DAQ) with Python	243
38.1 Introduction to DAQ	243
38.2 Data Acquisition using NI DAQ Devices	243
38.2.1 NI-DAQmx	245
38.2.2 Measurement Automation Explorer (MAX)	246
38.3 NI-DAQmx Python API	246
38.3.1 Analog Write	247
38.3.2 Analog Read	247
38.3.3 Digital Write	249
38.3.4 Digital Read	249
38.4 Controlling LEDs	250
38.5 Read Data from Temperature Sensors	252
38.5.1 Read Data from TMP36 Temperature Sensor	252
38.5.2 Read Data from Thermistor	256
38.5.3 Read Data NI TC-01 Thermocouple Device	260
38.6 Data Logging	261

VIII	Python Database Development	262
39	Database Applications with Python	263
39.1	Structured Query Language (SQL)	263
39.2	SQL Server	264
39.3	MySQL	264
39.4	MongoDB	264
40	MongoDB with Python	265
40.1	Introduction to MongoDB	265
40.2	MongoDB with Python	265
40.2.1	PyMongo	265
40.3	Additional Resources	266
IX	Python Application Development	267
41	Development of Applications with Python	268
41.1	Mathematics, Science and Engineering	269
41.2	Desktop GUI Applications	269
41.2.1	PyQt	270
41.2.2	PySide2	271
41.2.3	Tkinter	271
41.2.4	WxPython	271
41.3	Web Applications	272
41.4	Database Applications	272
41.4.1	SQL Server	272
41.4.2	MySQL	272
41.4.3	MariaDB	273
41.4.4	MongoDB	273
42	Python Integration with Visual Studio	274
43	Python Integration with LabVIEW	275
43.1	What is LabVIEW?	275
43.2	Using Python in LabVIEW	275
44	Raspberry Pi and Python	280
44.1	What is Raspberry Pi?	280
45	Machine Learning with Python	281
45.1	Introduction to Machine Learning	281
X	Resources	282
46	Python for MATLAB Users	283
46.1	Use Python inside MATLAB	284
46.2	Calling MATLAB from Python	285

47 Python Resources	287
47.1 Python Distributions	287
47.2 Python Libraries	287
47.3 Python Editors	287
47.4 Python Tutorials	288
47.5 Python in Visual Studio	288
 XI Solutions to Exercises	 291

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].

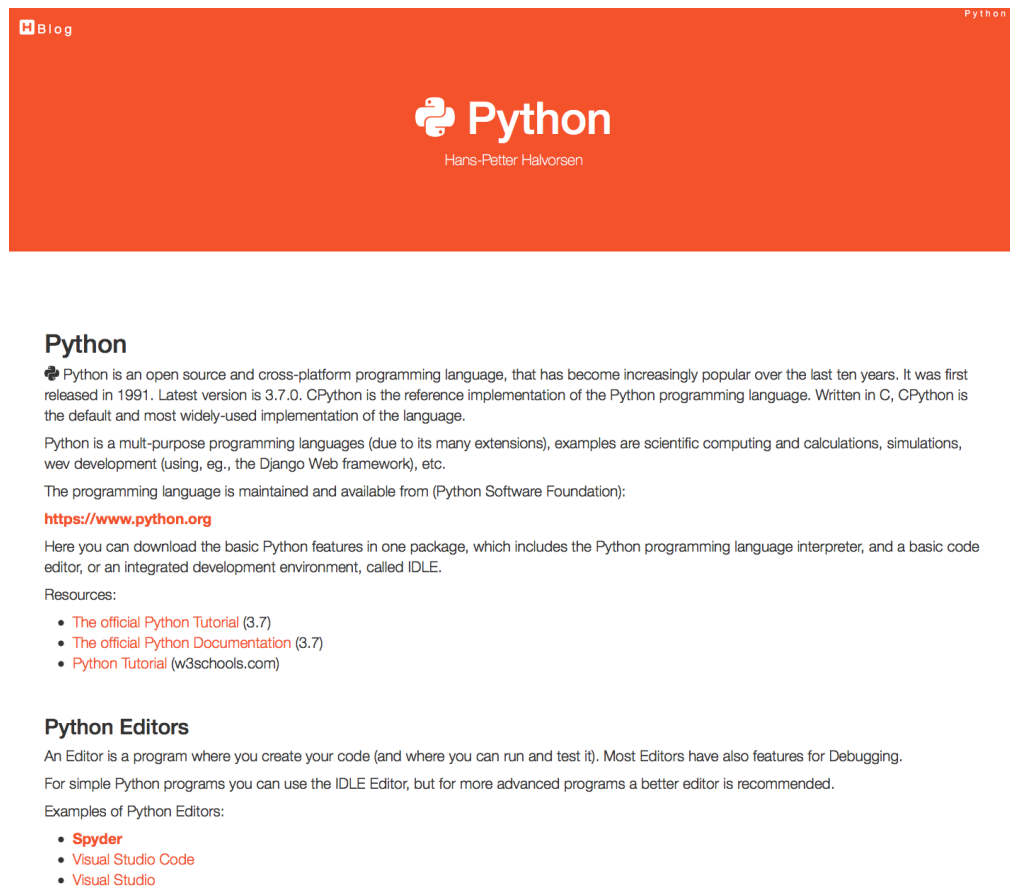


Figure 1.1: Web Site - Python

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

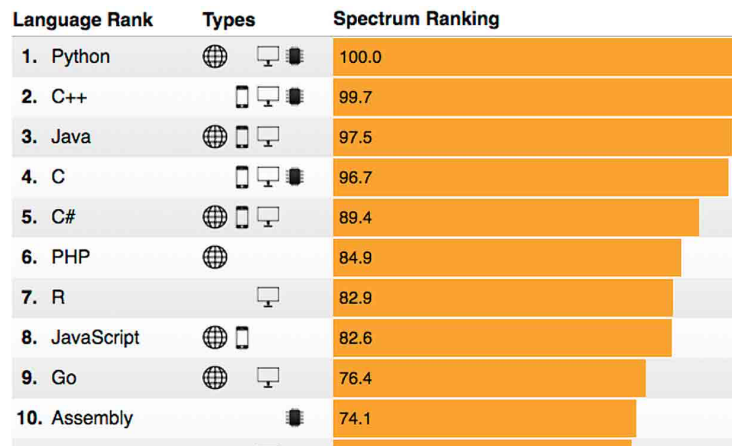


Figure 1.2: The Most Popular Programming Languages

- 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.

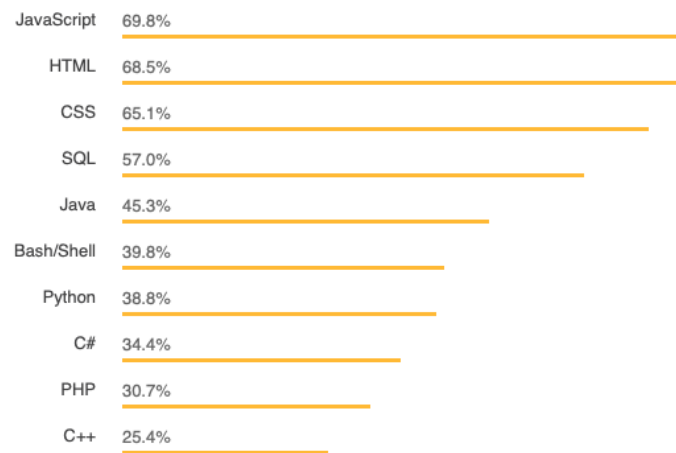


Figure 1.3: The Top Programming Languages - Stack Overflow Survey

- 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)

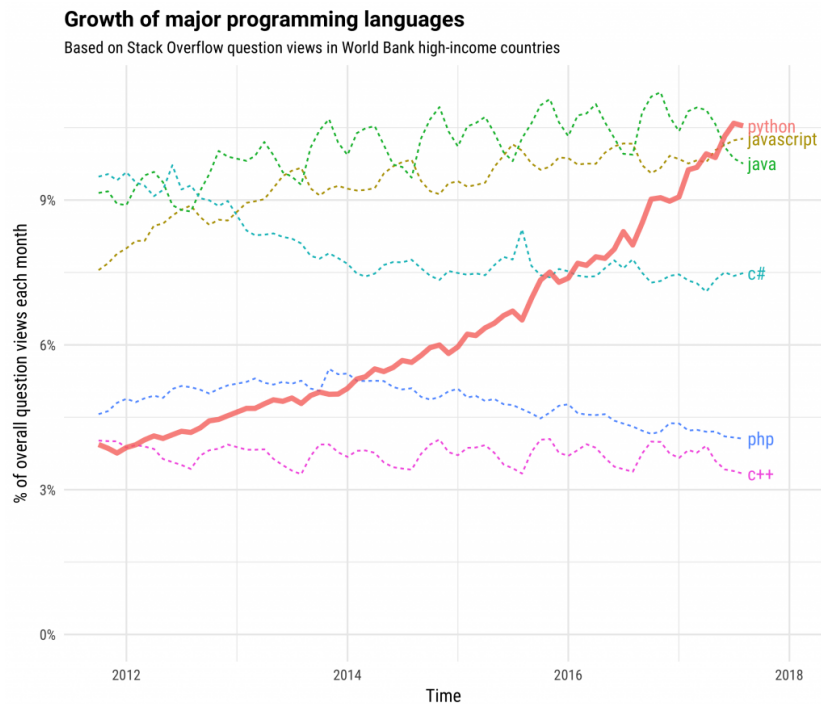


Figure 1.4: The Incredible Growth of Python

- 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/>

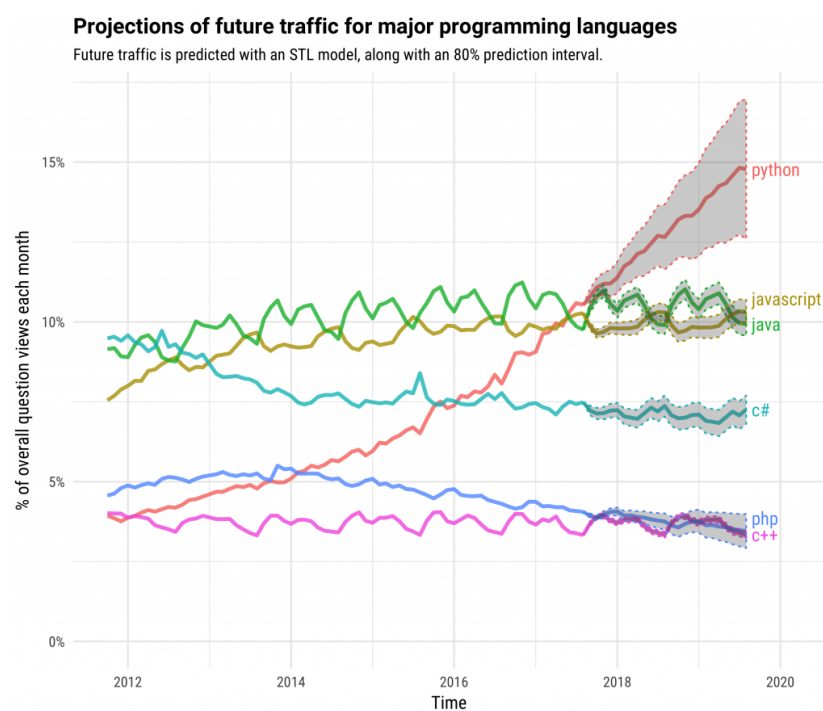


Figure 1.5: The Future of Python

Chapter 2

What is Python?

2.1 Introduction to Python

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

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

Python Home Page [6]:
<https://www.python.org>

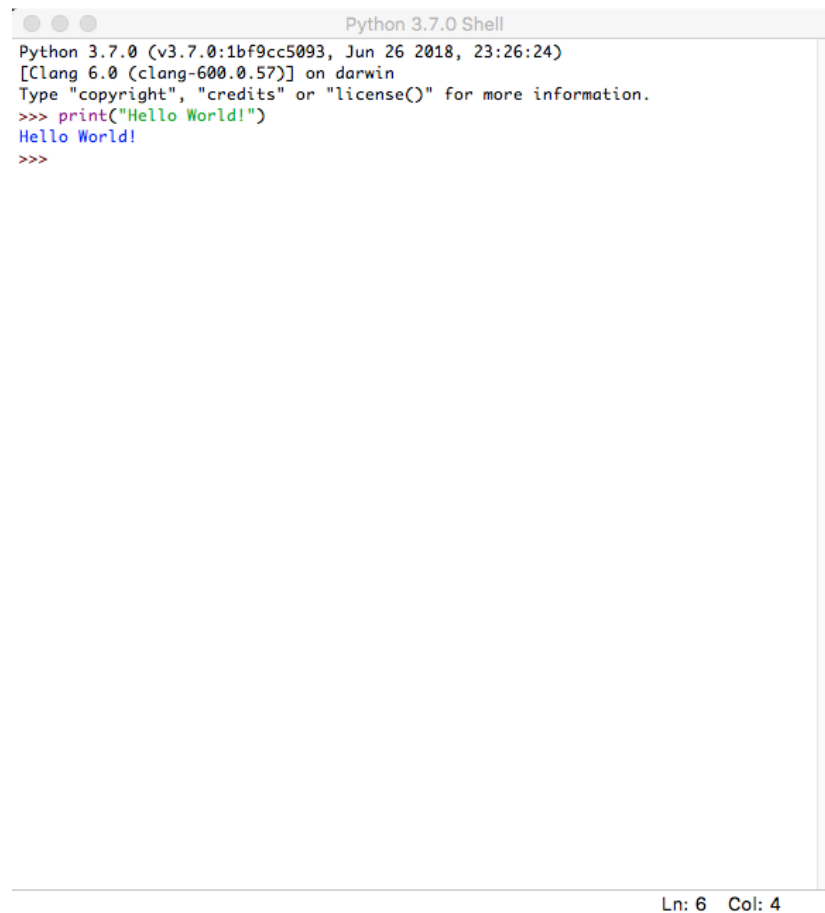
The programming language is maintained and available from (Python Software Foundation): <https://www.python.org> Here you can download the basic Python features in one package, which includes the Python programming language interpreter, and a basic code editor, or an integrated development environment,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

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

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

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

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



```
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!
>>>
```

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 your are going to develop in Python, etc.

2.4.1 Python IDLE

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

Web:
<https://www.python.org>

2.4.2 Visual Studio Code

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

Web:

<https://code.visualstudio.com>

Resources: Getting Started with Python in Visual Studio Code

2.4.3 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

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

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

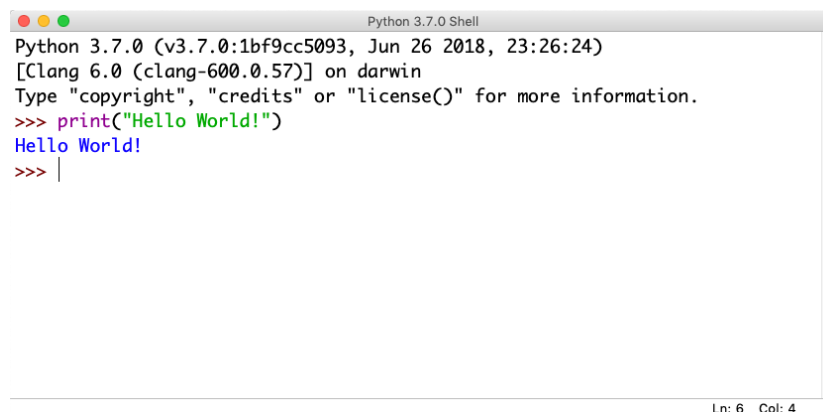
Start using Python

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 a macOS-style window titled "Python 3.7.0 Shell". The window contains 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.", and a prompt ">>>" followed by the command "print('Hello World!)" in a monospaced font. The output "Hello World!" is displayed below the command. The prompt ">>>" is followed by a vertical cursor bar. At the bottom right of the window, the status bar shows "Ln: 6 Col: 4".

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

Figure 3.1: Python Shell / Python IDLE Editor

3.2 My first Python program

We will start using Python and create some code examples.

Example 3.2.1. Plotting in Python

Lets open your Python Editor and type the following:

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

Listing 3.1: Hello World Python Example

[End of Example]

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

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

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

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

3.3 Python Shell

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

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

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

3.4 Running Python from the Console

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

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

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

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.

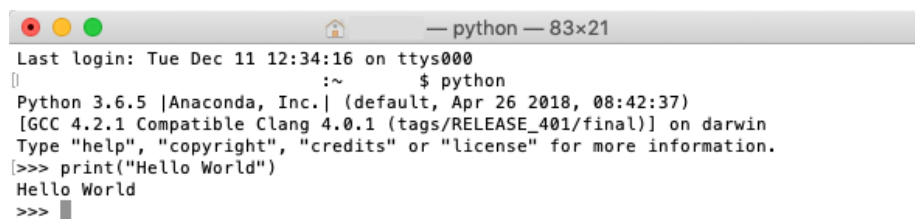
```
1 Last login: Tue Dec 11 12:34:16 on ttys000
2 Hans-Petter-Work-MacBook-Air:~ hansha$ python
3 Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)
4 [GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on
  darwin
5 Type "help", "copyright", "credits" or "license" for more
  information.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

See also Figure 3.2.



```
python — 83x21
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

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

You can also search for it from the start menu.

It should look like:

```
C:\Users\myusername>
```

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

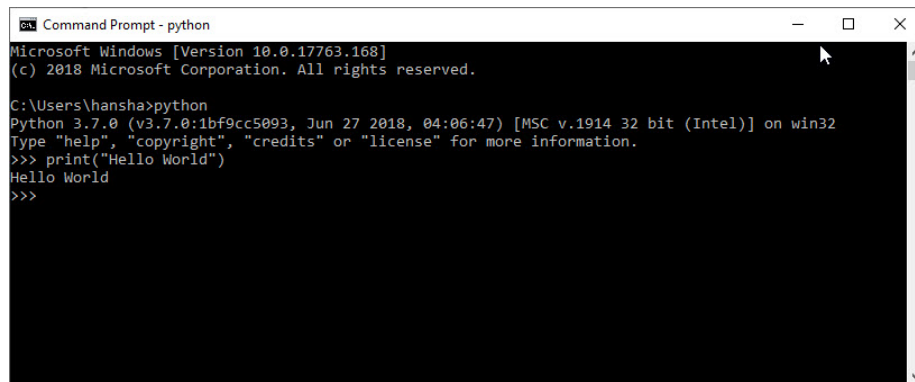


Figure 3.3: Command Prompt Windows

If you get an error message like this:

'python' is not recognized as an internal or external command, operable program or batch file.

Then you need to add Python to your path. See instructions below.

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

3.4.3 Add Python to Path

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

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

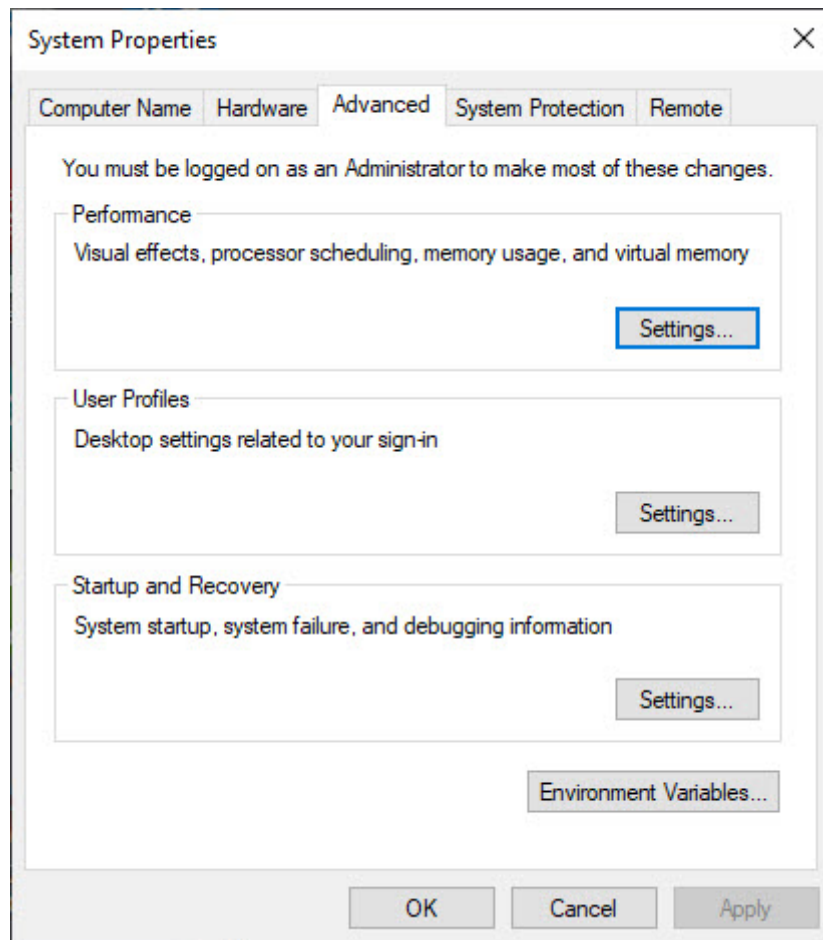


Figure 3.4: Windows System Properties

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

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

The Default Location is:

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

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

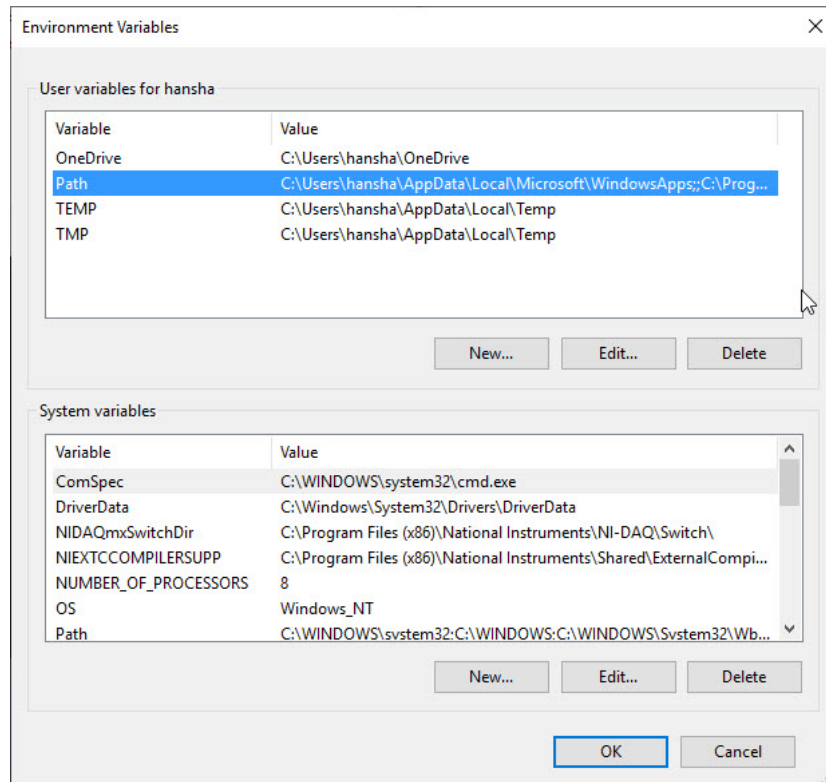


Figure 3.5: Windows System Properties

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.

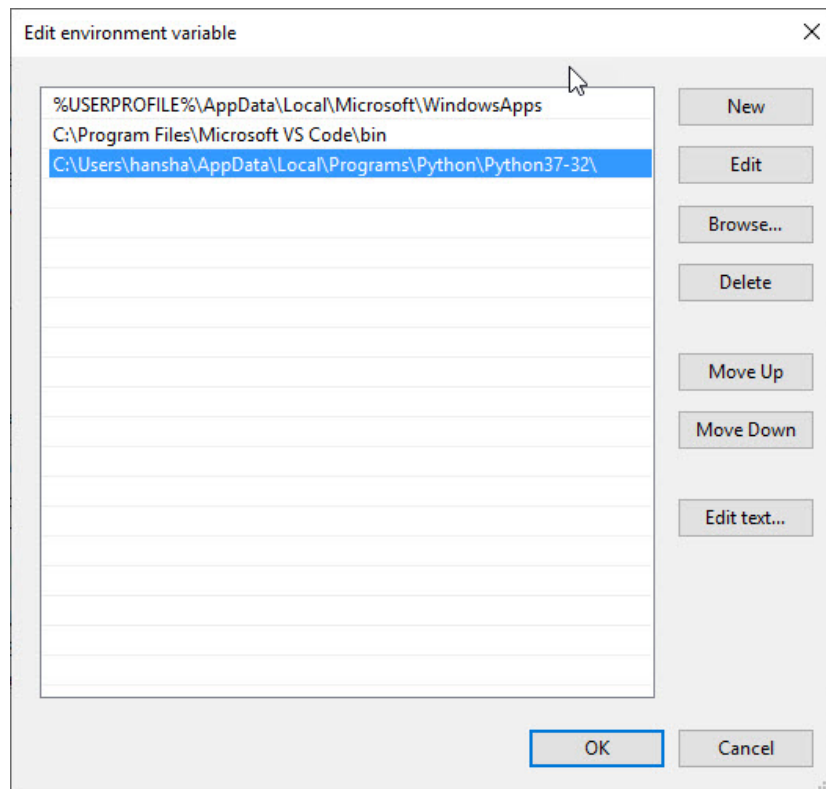


Figure 3.6: Windows System Properties

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:

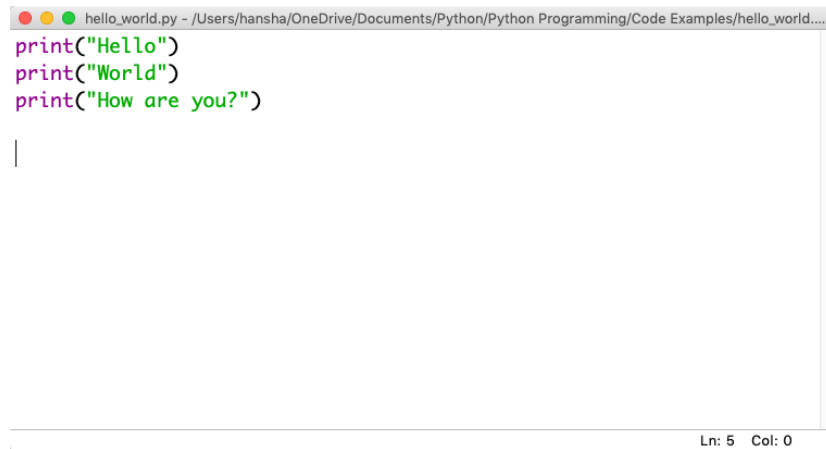
```
1 $ cd /Users/username/Downloads
2 $ python helloworld.py
```

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)!

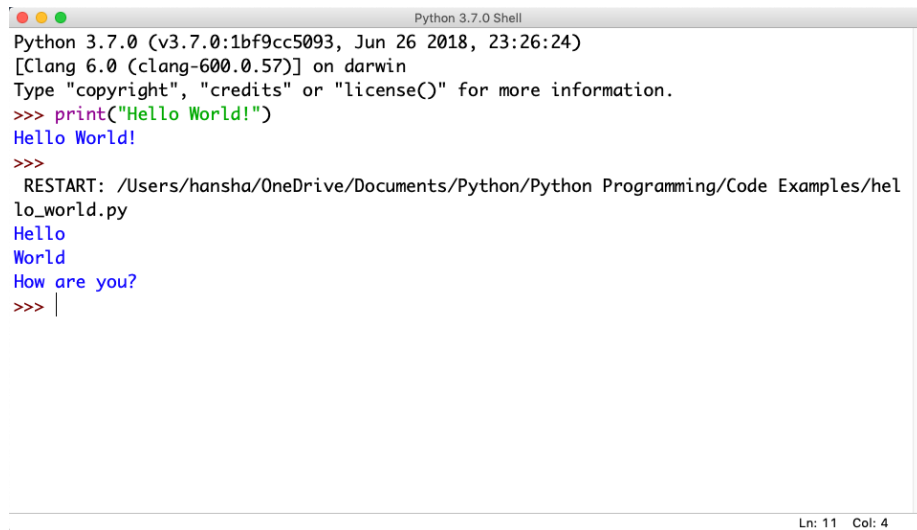
See also Figure 3.10.
Then it responds with:

```
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.

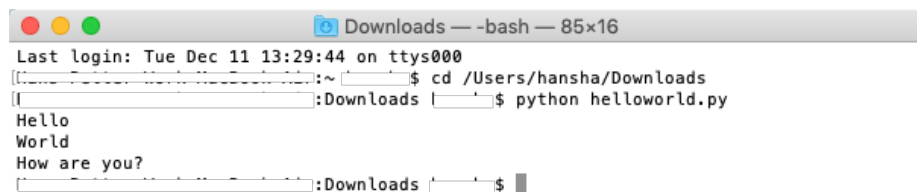
In the Spyder editor we have the Script Editor to the left and the interactive Python Shell or the Console window to the right. See See 3.11.

A screenshot of a macOS terminal window titled "Python 3.7.0 Shell". The window shows the output of running a Python script. The text inside the terminal is: 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? >>> | The status bar at the bottom right indicates "Ln: 11 Col: 4".

```
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?
>>> |
```

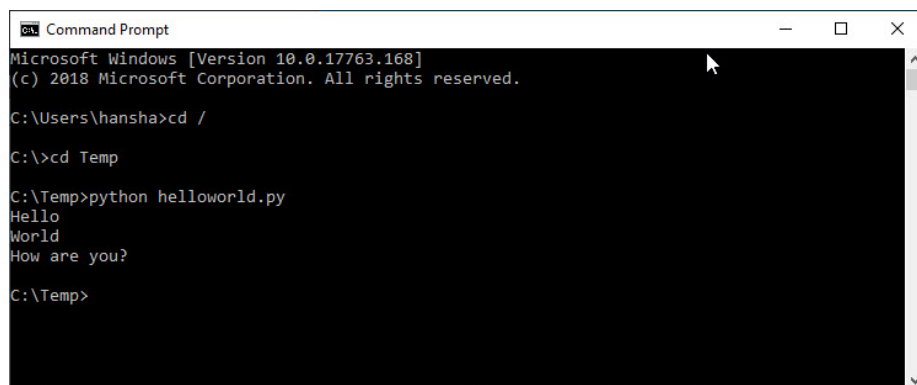
Ln: 11 Col: 4

Figure 3.8: Running a Python Script

A screenshot of a macOS terminal window titled "Downloads — -bash — 85x16". The window shows the output of running a Python script. The text inside the terminal is: Last login: Tue Dec 11 13:29:44 on ttys000 [~]\$ cd /Users/hansha/Downloads [Downloads]\$ python helloworld.py Hello World How are you? [Downloads]\$ The status bar at the bottom right indicates "Ln: 11 Col: 4".

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

Figure 3.9: Running Python Scripts from Console window on macOS

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the output of running a Python script. The text inside the terminal is: 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> The status bar at the bottom right indicates "Ln: 11 Col: 4".

```
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

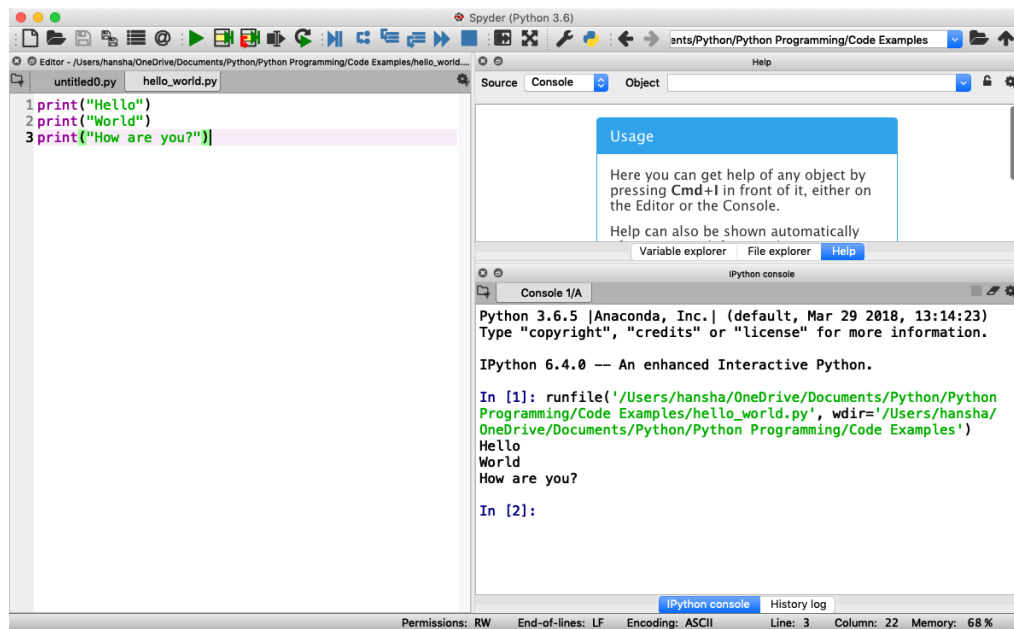


Figure 3.11: Running a Python Script in Spyder

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.

Python

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 implementing 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

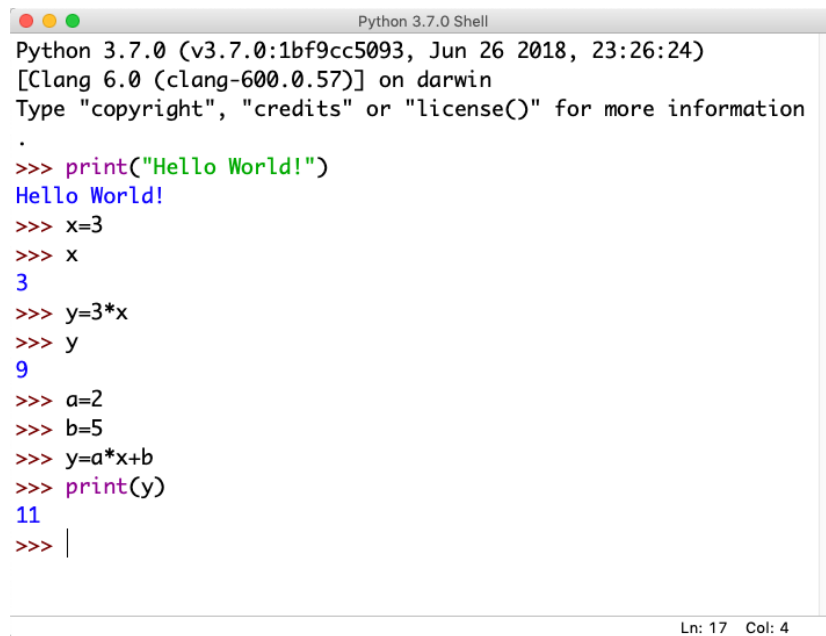
As seen in the examples, you can use the *print()* command in order to show the values on the screen.

[End of Example]

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

You don't need to define the variables before you use them (like you need to in, e.g., C/C++/C).

Figure 4.1 shows these examples using the basic IDLE editor.

A screenshot of a Python 3.7.0 Shell window. The window title is "Python 3.7.0 Shell". The text inside shows the Python version and build information: "Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24) [Clang 6.0 (clang-600.0.57)] on darwin". It then prompts the user to type "copyright", "credits" or "license()" for more information. The user has entered a period ".". The prompt is ">>>". The user has entered "print('Hello World!')". The output is "Hello World!". The prompt is ">>>". The user has entered "x=3". The prompt is ">>>". The user has entered "x". The output is "3". The prompt is ">>>". The user has entered "y=3*x". The prompt is ">>>". The user has entered "y". The output is "9". The prompt is ">>>". The user has entered "a=2". The prompt is ">>>". The user has entered "b=5". The prompt is ">>>". The user has entered "y=a*x+b". The prompt is ">>>". The user has entered "print(y)". The output is "11". The prompt is ">>>". The cursor is at the end of the line. The status bar at the bottom right shows "Ln: 17 Col: 4".

```
Python 3.7.0 Shell
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information
.
>>> print('Hello World!')
Hello World!
>>> x=3
>>> x
3
>>> y=3*x
>>> y
9
>>> a=2
>>> b=5
>>> y=a*x+b
>>> print(y)
11
>>> |
```

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
```

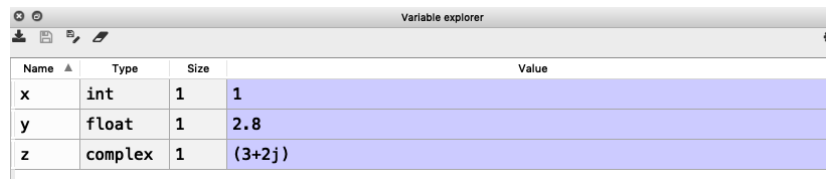
Listing 4.5: Numeric Types in Python

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. Plotting 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. Plotting 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 π , e , ∞ , nan , etc. 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. An 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:
<http://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 to arrays with data. We want to plot `x` vs. `y`. We can assume `x` is a time series and `y` is the corresponding temperature i 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 following plot:

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.

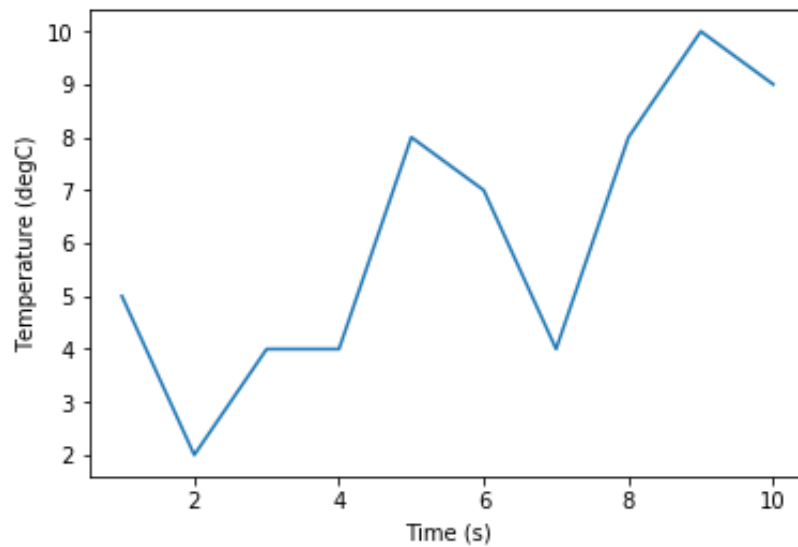


Figure 4.3: Plotting in Python

[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:

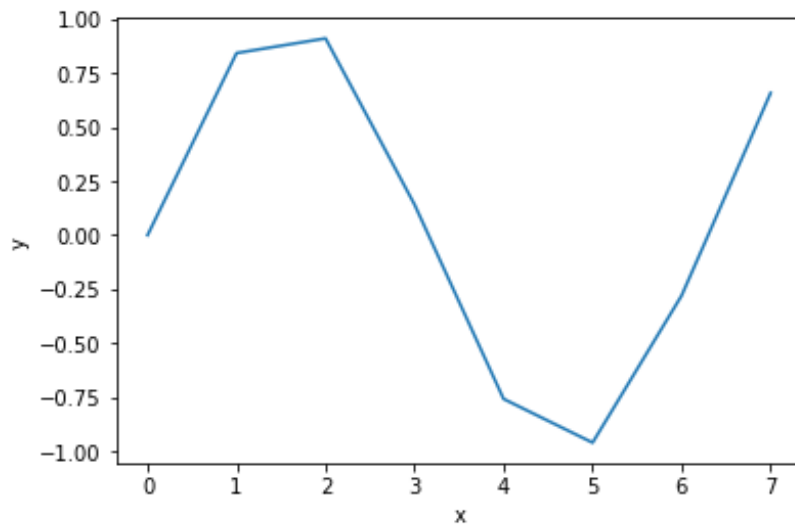


Figure 4.4: Plotting a Sine function in Python

```

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

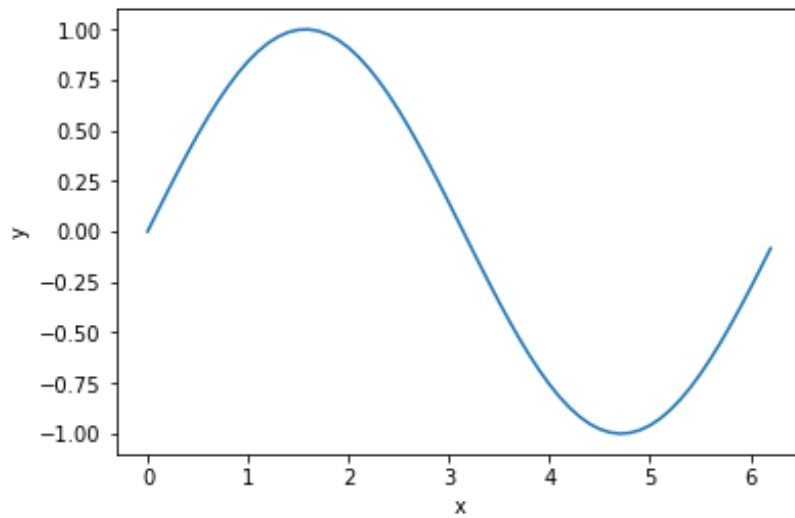


Figure 4.5: Plotting a Sine function in Python - Better Implementation

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
12 z = np.cos(x)
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]

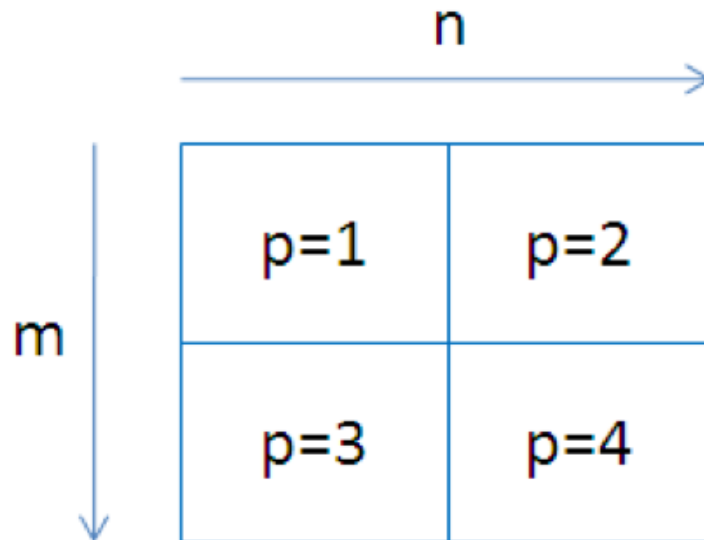


Figure 4.6: Creating Subplots in Python

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 has 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 For Loops in Python

```
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: Using Arrays in Python

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: Using Arrays in Python

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: Using Arrays in Python

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. Using For Loops in Python

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

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

Where:

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

where T is the time constant.

The solution for the differential equation is:

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

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. Create a Function in a separate File

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  
3     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  
3     totalsum = 0  
4  
5     #Find the Sum of all the numbers  
6     for x in data:  
7         totalsum = totalsum + x  
8  
9  
10    #Find the Mean or Average of all the numbers  
11  
12    N = len(data)  
13  
14    mean = totalsum/N  
15  
16  
17    return totalsum , mean  
18  
19  
20
```

```

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 **calcoverage** 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:

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

This gives:

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

and

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

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     <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
6 car = Car()
7
8
9 print(car.model)
10 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)
10
11
```

```

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 # Lets start using the Class
12 car1 = Car("Tesla", "Red")
13
14 car1.displayCar()
15
16 car2 = Car("Ford", "Green")
17
18 print(car2.model)
19 print(car2.color)
20
21 car3 = Car("Volvo", "Blue")
22
23 print(car3.model)
24 print(car3.color)
25
26 car3.color="Black"
27
28 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. Now 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
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 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:

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

This gives:

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

and

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

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", "w")
2
3 data = "Helo World"
4
5 f.write(data)
6
7 f.close()
```

Listing 9.1: Write Data to a File

[End of Example]

9.3 Read Data from a File

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

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

Example 9.3.1. Read Data from a File

```
1 f = open("myfile.txt", "r")
2
3 data = f.read()
4
5 print(data)
6
7 f.close()
```

Listing 9.2: Read Data from a File

[End of Example]

9.4 Logging Data to File

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

Example 9.4.1. Logging Data to File

```

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

```

Listing 9.3: Logging Data to File

[End of Example]

Example 9.4.2. Read Logged Data from File

```

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

```

Listing 9.4: Read Logged Data from File

[End of Example]

9.5 Web Resources

Below you find different useful resources for File Handling.

Python File Handling - w3school:

https://www.w3schools.com/python/python_file_handling.asp

Reading and Writing Files - python.org:

<https://docs.python.org/3/tutorial/inputoutput.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]

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

Table 9.1: Logged Data

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
- 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 need different versions of Python or different versions 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.

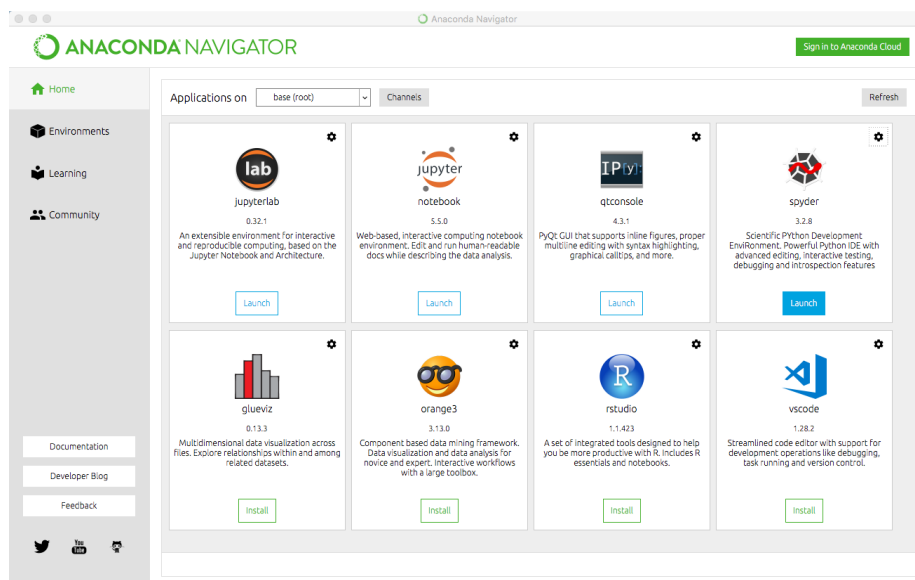


Figure 14.1: Anaconda Navigator

Chapter 15

Enthought Canopy

Enthought Canopy is a Python Platform or Python Distribution for Scientists and Engineers.

It is available for Windows, macOS and Linux.

Canopy is freely available to all users under the Canopy license. Canopy provides access to several hundreds Python packages, including NumPy, SciPy, Pandas, Matplotlib, and IPython.

In addition, we have the Canopy Python Editor.

Enthought Canopy is a competitor to the Anaconda Python Distribution. It is a matter of taste who you prefer.

Web:

<https://www.enthought.com/product/canopy/>

Part IV

Python Editors

Chapter 16

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 your 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 17

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.

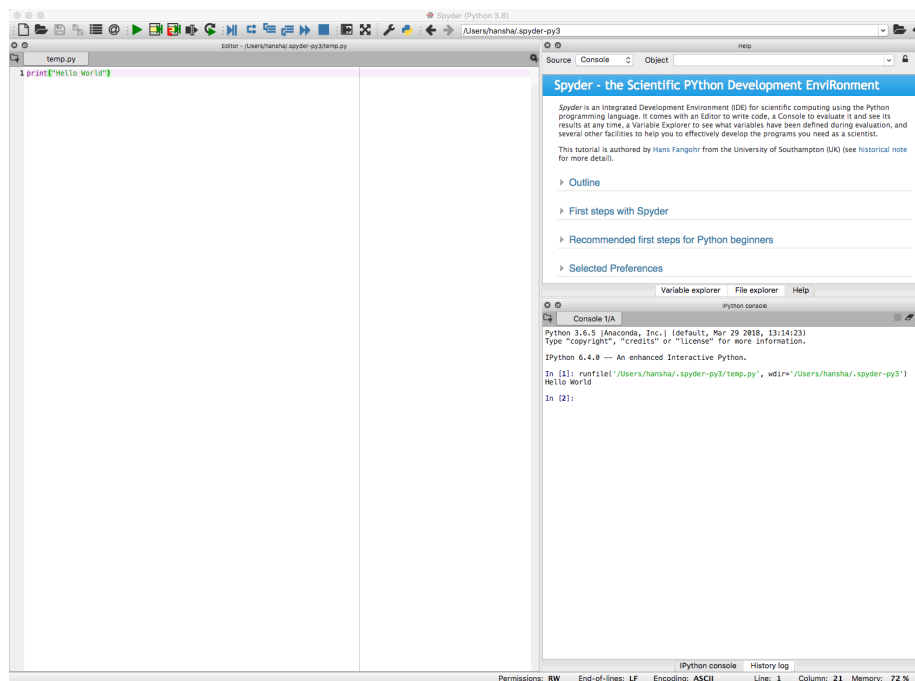


Figure 17.1: Spyder Editor

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>

Chapter 18

Visual Studio Code

18.1 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.

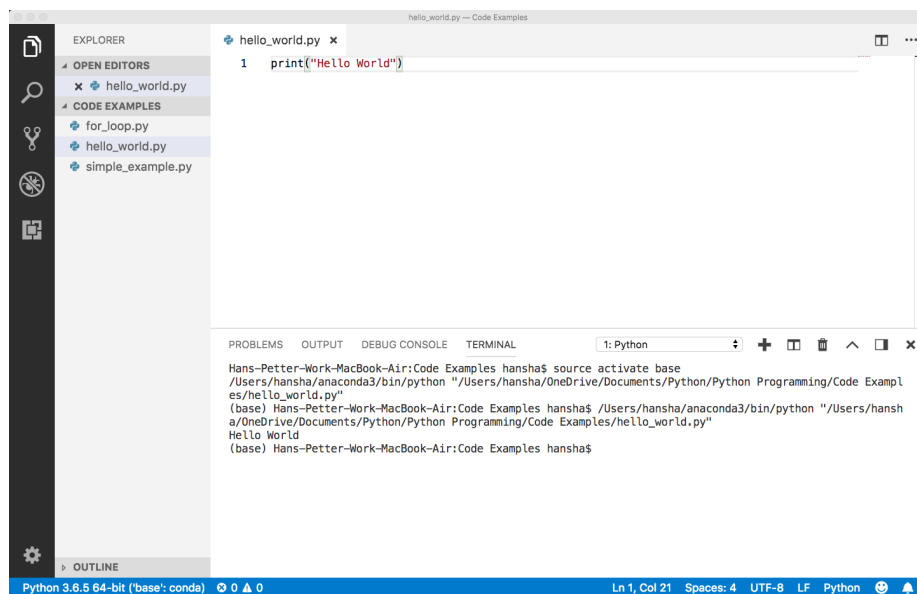


Figure 18.1: Using Visual Studio Code as Python Editor

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

Web:

<https://code.visualstudio.com>

Wikipedia:

<https://en.wikipedia.org/wiki/VisualStudioCode>

18.2 Python in Visual Studio Code

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

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

<https://www.python.org>

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

Web:

<https://code.visualstudio.com/docs/languages/python>

Chapter 19

Visual Studio

19.1 Introduction to 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.

You could say Visual Studio is the big brother of Visual Studio Code.

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

Go to my Web Site to learn more about Visual Studio and C programming:

<https://www.halvorsen.blog/>

Visual Studio and C:

<https://www.halvorsen.blog/documents/programming/csharp/>

19.2 Work with Python in Visual Studio

Work with Python in Visual Studio:

<https://docs.microsoft.com/visualstudio/python/>

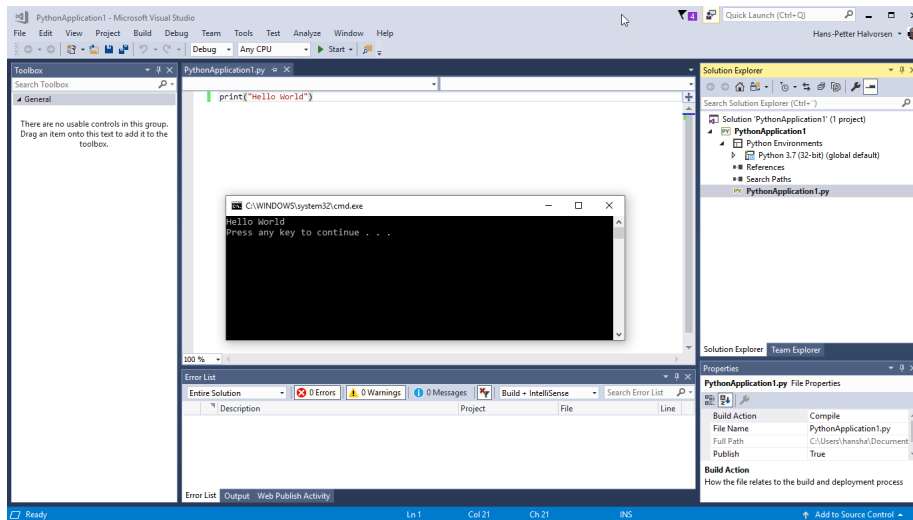


Figure 19.1: Using Visual Studio as Python Editor

19.2.1 Make Visual Studio ready for Python Programming

Visual Studio is mainly for Windows. A MacOS version of Visual Studio does exist, but it has a lot less features than the Windows edition.

Note that Python support is available only on Visual Studio for Windows. If you use Mac and Linux, you need to use Visual Studio Code. You could say Visual Studio Code is a down-scaled version of Visual Studio.

Visual Studio (from 2017), has integrated support for Python, it is called "Python Support in Visual Studio". Even if it is integrated, you need to manually select which components you want to install on your computer. Make sure to download and run the latest Visual Studio 2017 installer for Windows.

When you run the Visual Studio installer (either for the first time or if you already have installed Visual Studio 2017 and want to modify it) the window shown in Figure 19.2 pops up.

The installer presents you with a list of so-called workloads, which are groups of related options for specific development areas. For Python, select the "Python development" workload and select Install (Figure 19.3).

19.2.2 Python Interactive

To quickly test Python support, launch Visual Studio, press Alt+I (or select from the menu: Tools - Python - Python Interactive Window) to open the Python Interactive window. See Figure 19.4.

Let's write something like this:

```
1 >>> a = 2
```

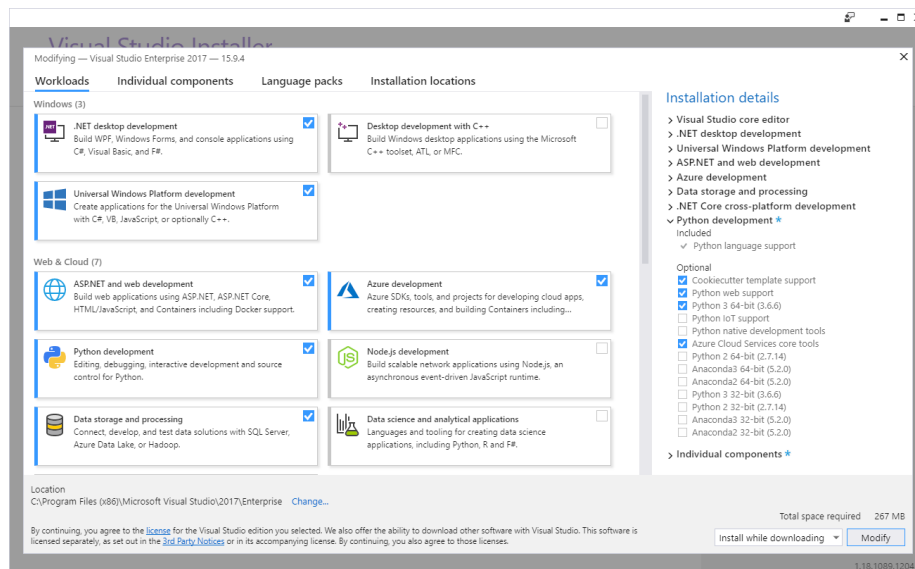


Figure 19.2: Installing Python Extension for Visual Studio

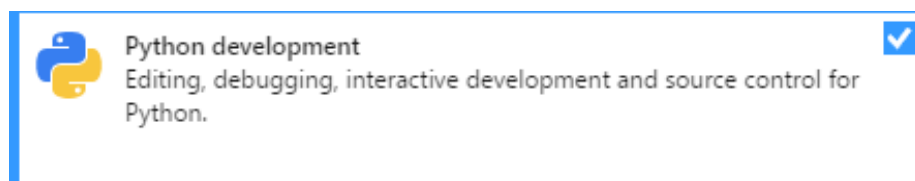


Figure 19.3: Python Development Workload

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

19.2.3 New Python Project

Lets see how we can create a Python Application.

Start by select from the menu: File - New - Project... The New Project window pops up. See Figure 19.5.

We can create an ordinary Python Application (one or more Python Scripts), we can choose to create a Web Application using either Web Frameworks like Django or Flask, or we can create different Desktop GUI applications. We can also create Games.

Example 19.2.1. Python Hello World Application in Visual Studio

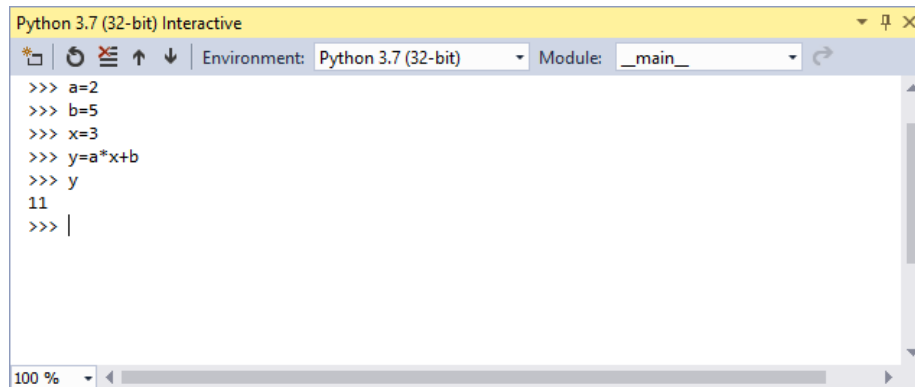


Figure 19.4: Python Interactive

We start by creating a basic Hello World Python Application. See Figure 19.1. Select File - New - Project... The New Project window pops up. See Figure 19.5.

Name the project, e.g, "PythonApplication1".

In the Project Explorer, open the "PythonApplication1.py" file and enter the following Python code:

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

Hit F5 (or click the green arrow) in order to run or execute the Python program. You can also right click on the file and select "Start without Debugging".

[End of Example]

Example 19.2.2. Visual Studio Python Plotting

Create a new Python File by right click in the Solution Explorer and select Add - New Item... and then select "Empty Python File".

Enter the following Python Code:

```
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.title('y=sin(x)')
```

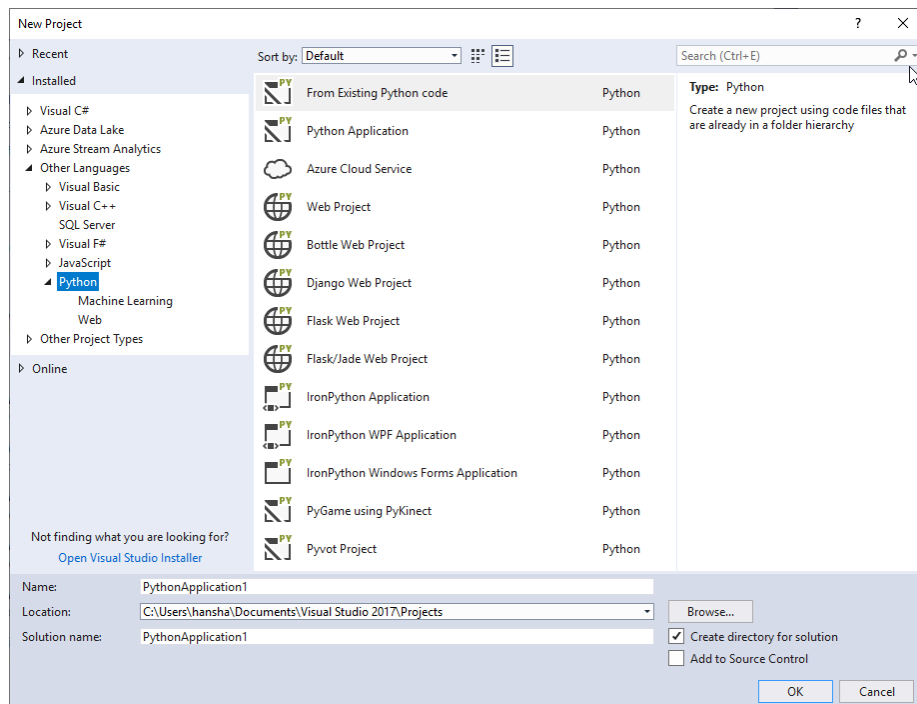


Figure 19.5: New Python Project

```

14 plt.xlabel('x')
15 plt.ylabel('y')
16 plt.grid()
17 plt.axis([0, 2*np.pi, -1, 1])
18 plt.show()

```

See also Figure 19.6.

Make sure to select proper Python Environment. See Figure (19.7). Visual Studio supports multiple Python Environments.

In this example we use the Matplotlib package for plotting, so we need to have that package installed on the computer. You can install the Matplotlib package in different Python Environments.

I have installed the Matplotlib package as part of the Anaconda distribution setup, so I select "Anaconda x.x.x" in the Python Environments window.

If you haven't installed the Matplotlib package yet (either as part of Anaconda or manually using PIP), you can also easily install Python packages from Visual studio. See Figure 19.8.

You can also easily see which Python Packages that are installed for the different Python Environments. See Figure 19.9.

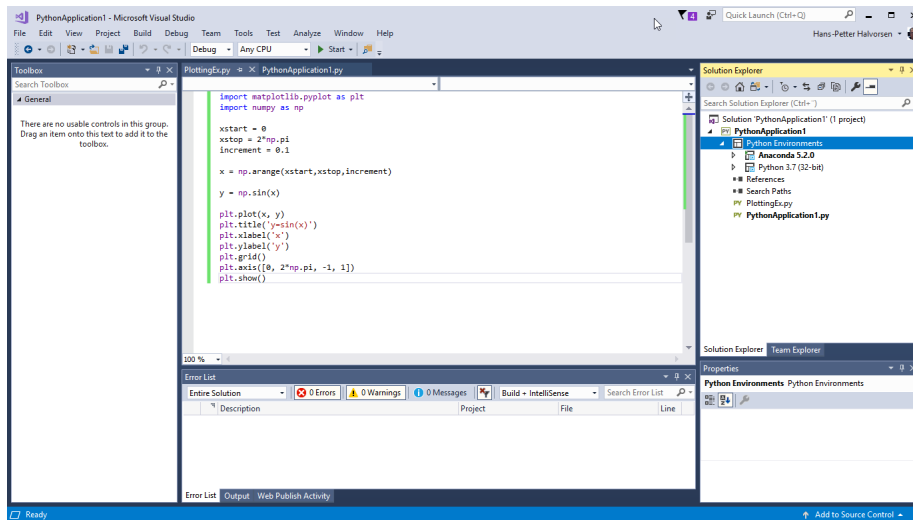


Figure 19.6: Python Plotting Example with Visual Studio

The good thing about using Visual Studio is that you have a graphical user interface for everything, you don't need to use the Command window etc. for installing Python Packages, etc.

Hit F5 (or click the green arrow) in order to run or execute the Python program. You can also right click on the file and select "Start without Debugging".

We get the following results, see Figure 19.10.

[End of Example]

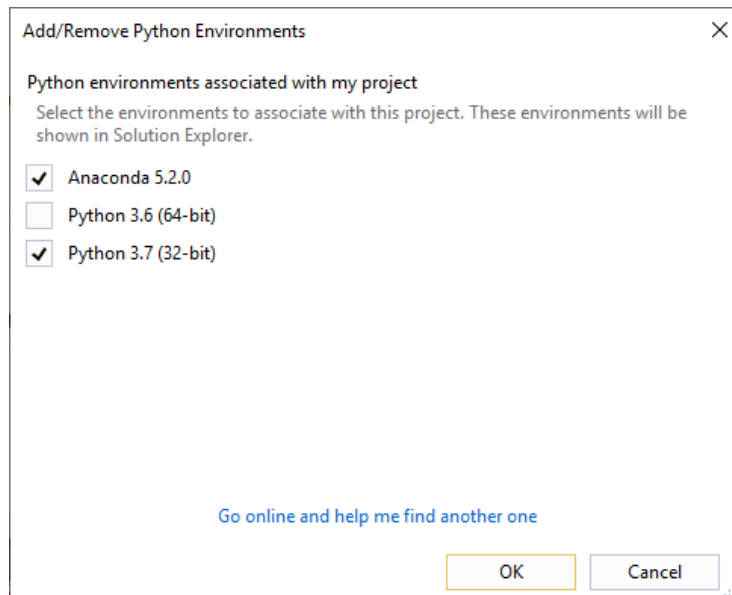


Figure 19.7: Select your Python Environment

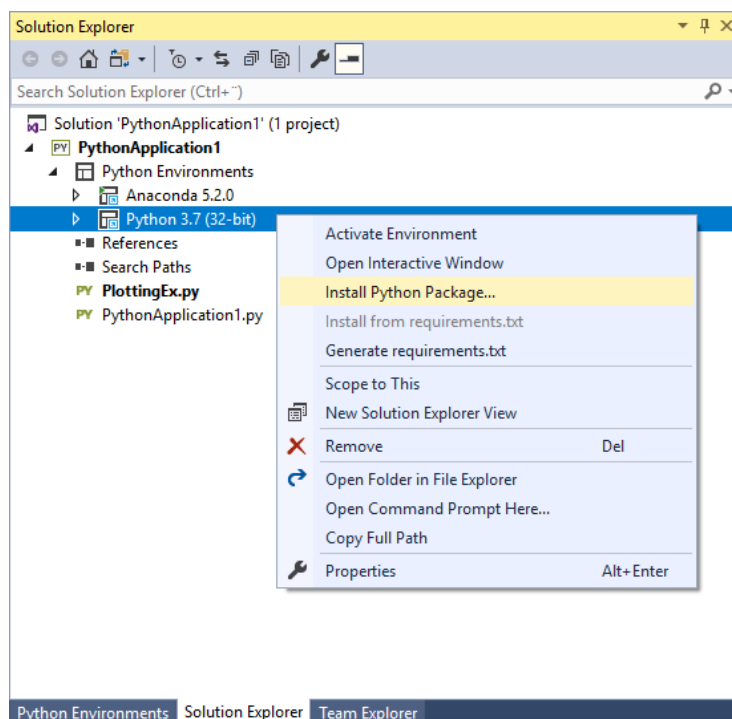


Figure 19.8: Install Python Packages from Visual Studio

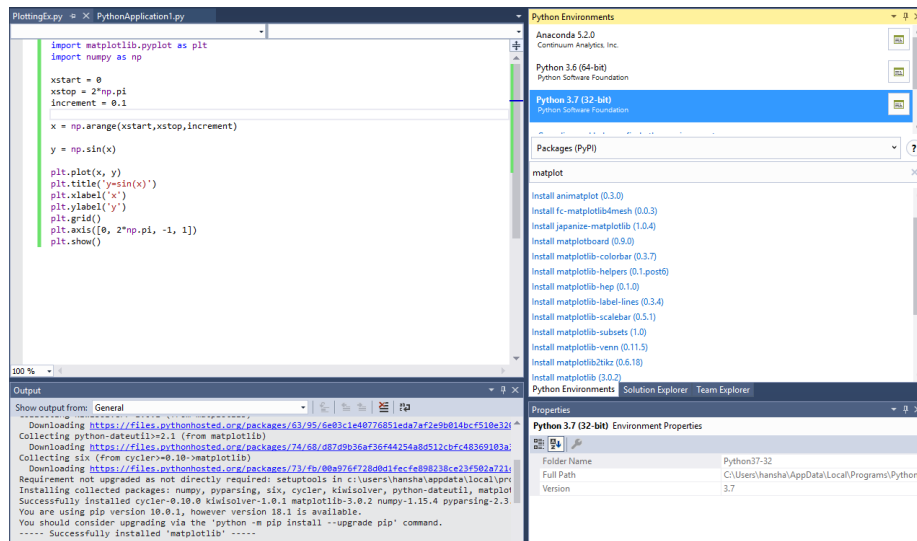


Figure 19.9: Installing Python Packages for different Python Environments from Visual Studio

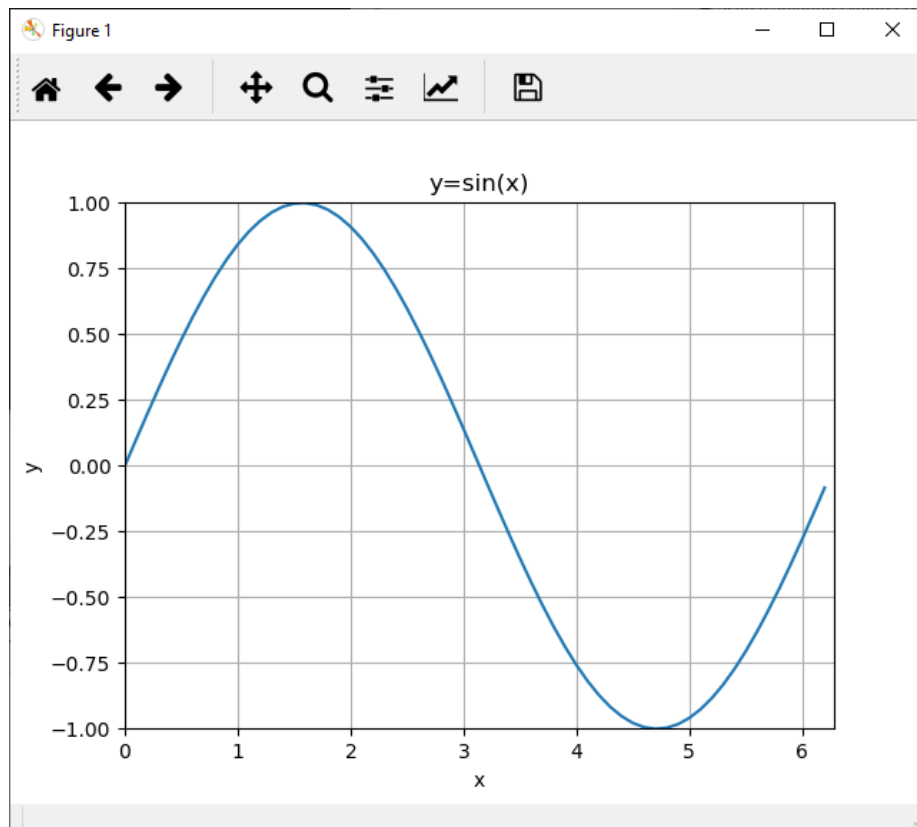


Figure 19.10: Python Plotting Example with Visual Studio

Chapter 20

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.

The PyCharm Editor is shown in Figure 20.1.

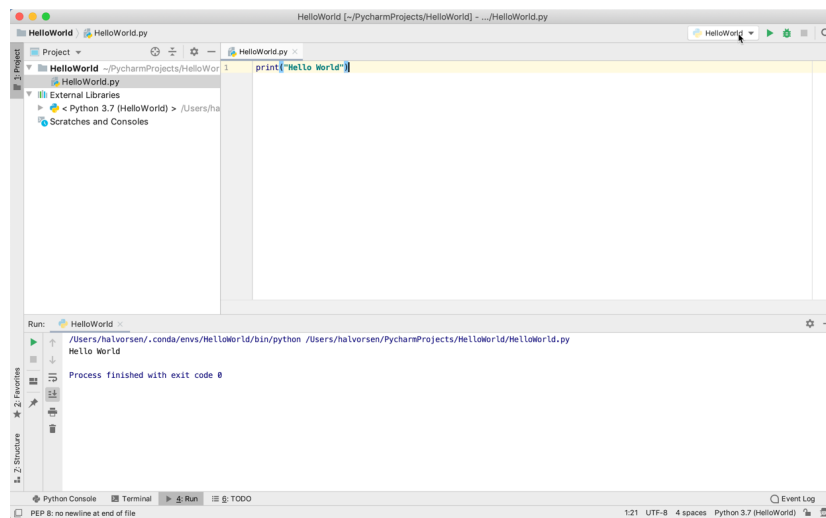


Figure 20.1: PyCharm Python Editor

Web:

<https://www.jetbrains.com/pycharm/>

Wikipedia:

<https://en.wikipedia.org/wiki/PyCharm>

Anaconda and JetBrains also have a collaboration and offer what they call PyCharm for Anaconda. You can download it here:

<https://www.jetbrains.com/pycharm/promo/anaconda/>

We have code editors like Visual Studio and Visual Studio Code which can be used for many different programming languages by installing different types of plugins.

Editors like Spyder and PyCharm are tailor-made editors for the Python language.

Spyder is light-weight IDE typically used for scientific use. PyCharm on the other hand is full-blown IDE for software development in general by using the Python language. It supports many plugins, it's easier to program Django, etc.

Chapter 21

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

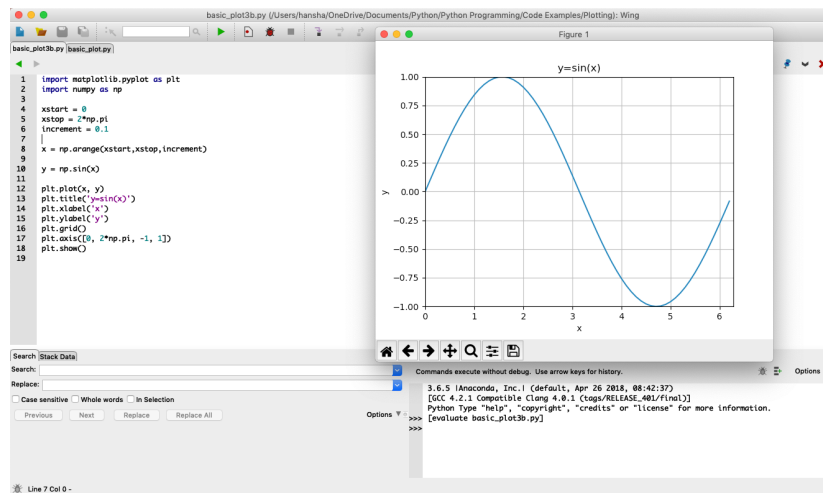


Figure 21.1: Wing Python IDE

Web:
<https://wingware.com>

Wikipedia:
https://en.wikipedia.org/wiki/Wing_I_DE

Chapter 22

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.

The Notebook has support for over 40 programming languages, including Python.

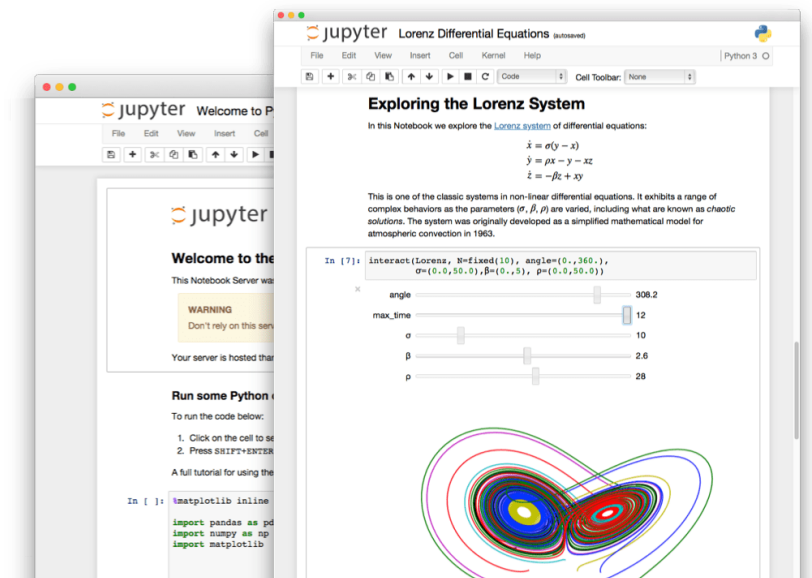


Figure 22.1: Jupyter Notebook [16]

Web:
<http://jupyter.org>

Wikipedia:
https://en.wikipedia.org/wiki/Project_Jupyter

22.1 JupyterHub

JupyterHub is a multi-user version of the notebook designed for companies, classrooms and research labs [17].

JupyterHub runs in the cloud or on your own hardware.

JupyterHub is open-source and designed to be run on a variety of infrastructure. This includes commercial cloud providers, virtual machines, or even your own laptop hardware.

Web:

<http://jupyter.org/hub>

22.2 Microsoft Azure Notebooks

Microsoft Azure Notebooks is a version of Jupyter Notebook from Microsoft.

The good thing about Microsoft Azure Notebooks is that you have the infrastructure and everything up and running ready for you to use. You can use it for free as well.

Web:

<https://notebooks.azure.com>

Example 22.2.1. Example Name

Figure 22.2 shows an overview of my Azure Notebook Projects.

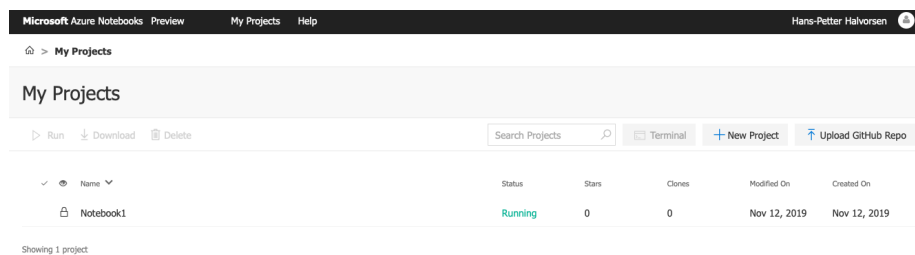


Figure 22.2: Azure Notebook Projects

Figure 22.3 shows an overview of my Azure Notebook Project Notebooks.

Figure 22.4 shows an example of a simple Notebook.

[End of Example]

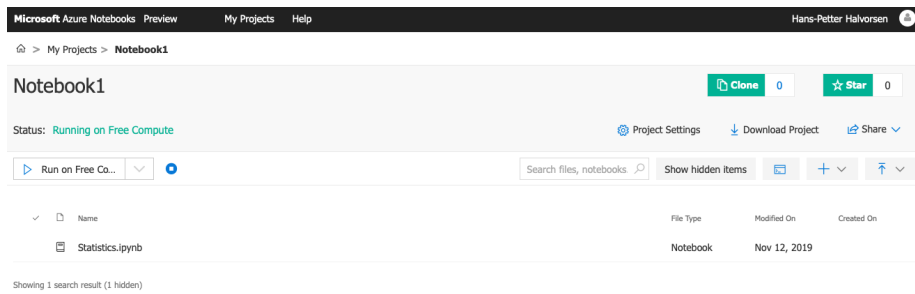


Figure 22.3: Azure Notebook Project Notebooks

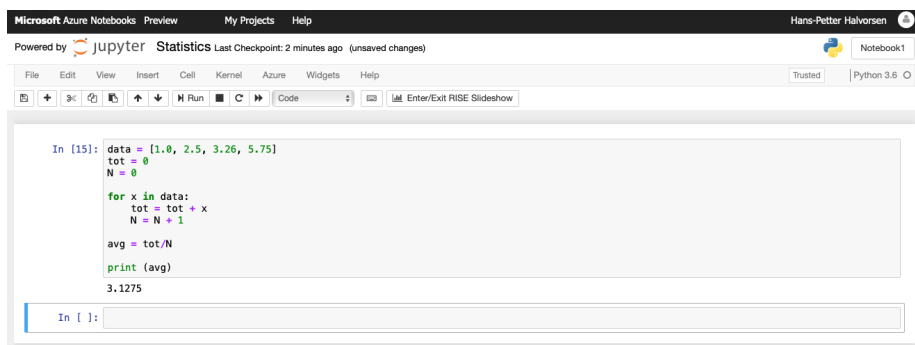


Figure 22.4: Azure Notebook Example

Part V

Python for Mathematics
Applications

Chapter 23

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/>

23.1 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 π , e , ∞ , nan , etc. etc.

Example 23.1.1. 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/>

23.1.1 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 23.1.1. Create Mathematical Expressions in Python

Create a function that calculates the following mathematical expression:

$$z = 3x^2 + \sqrt{x^2 + y^2} + e^{\ln(x)} \quad (23.1)$$

Test with different values for x and y.

[End of Exercise]

Exercise 23.1.2. Create advanced Mathematical Expressions in Python

Create the following expression in Python:

$$f(x) = \frac{\ln(ax^2 + bx + c) - \sin(ax^2 + bx + c)}{4\pi x^2 + \cos(x - 2)(ax^2 + bx + c)} \quad (23.2)$$

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 23.1.3. Pythagoras

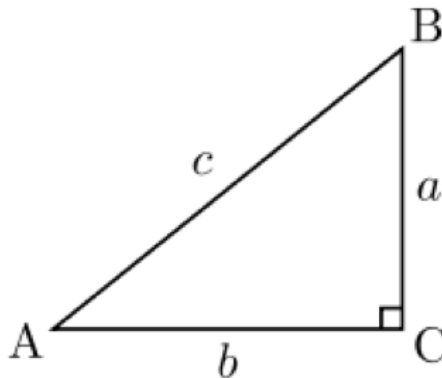


Figure 23.1: Right-angled triangle

Pythagoras theorem is as follows:

$$c^2 = a^2 + b^2 \quad (23.3)$$

Create a function that uses Pythagoras to calculate the hypotenuse of a right-angled triangle (Figure 23.1), e.g.:

```

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

```

[End of Exercise]

Exercise 23.1.4. Albert Einstein

Given the famous equation from Albert Einstein:

$$E = mc^2 \quad (23.4)$$

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 23.1.5. 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 ??.

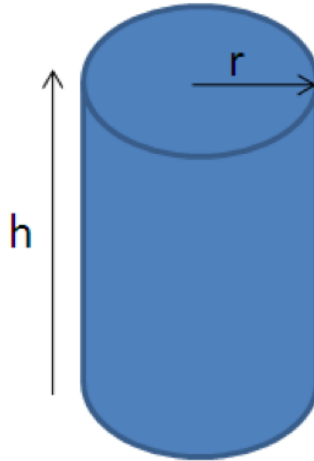


Figure 23.2: cylinder

[End of Exercise]

23.2 Statistics

23.2.1 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:

$$\bar{x} = \frac{x_1 + x_2 + x_3 + \dots + x_N}{N} = \frac{1}{N} \sum_{i=1}^N x_i \quad (23.5)$$

Example 23.2.1. Mean

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

Mean:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i = \frac{2.2 + 4.5 + 6.2 + 3.6 + 2.6}{5} = \frac{19.1}{5} = 3.82 \quad (23.6)$$

[End of Example]

Variance:

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

$$var(x) = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2 \quad (23.7)$$

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.

$$std(x) = \sigma = \sqrt{var} = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (23.8)$$

We typically use the symbol σ for standard deviation.

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

23.2.2 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 23.2.2. 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 23.1: 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 23.2.3. 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 23.2: Statistics using the NumPy Library

[End of Example]

Exercise 23.2.1. 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]

23.3 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 23.3.1. 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 23.3: 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 23.3.2. 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 23.4: 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 23.3.3. 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 23.5: Trigonometric Functions using NumPy

This Python script gives the plots as shown in Figure 23.3.

[End of Example]

Exercise 23.3.1. 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:

$$2\pi[\text{radians}] = 360[\text{degrees}] \quad (23.9)$$

This gives:

$$d[\text{degrees}] = r[\text{radians}] \times \left(\frac{180}{\pi}\right) \quad (23.10)$$

and

$$r[\text{radians}] = d[\text{degrees}] \times \left(\frac{\pi}{180}\right) \quad (23.11)$$

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 23.3.2. Trigonometric functions on right triangle

Given right triangle as shown in Figure 23.4.

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:

$$\sin(A) = \frac{a}{c} \rightarrow A = \arcsin\left(\frac{a}{c}\right) \quad (23.12)$$

$$\cos(A) = \frac{b}{c} \rightarrow A = \arccos\left(\frac{b}{c}\right) \quad (23.13)$$

$$\tan(A) = \frac{a}{b} \rightarrow A = \arctan\left(\frac{a}{b}\right) \quad (23.14)$$

We may also need to use the Pythagoras’ theorem:

$$c^2 = a^2 + b^2 \quad (23.15)$$

```
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 23.3.3. Law of Cosines

Given the triangle as shown in Figure 23.5.

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

$$c^2 = a^2 + b^2 - 2ab \cos(C) \quad (23.16)$$

Test the functions to make sure it works properly.

[End of Exercise]

Exercise 23.3.4. 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]

23.4 Polynomials

A polynomial is expressed as:

$$p(x) = p_1x^n + p_2x^{n-1} + \dots + p_nx + p_{n+1} \quad (23.17)$$

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/polynomial_class_in_python.php

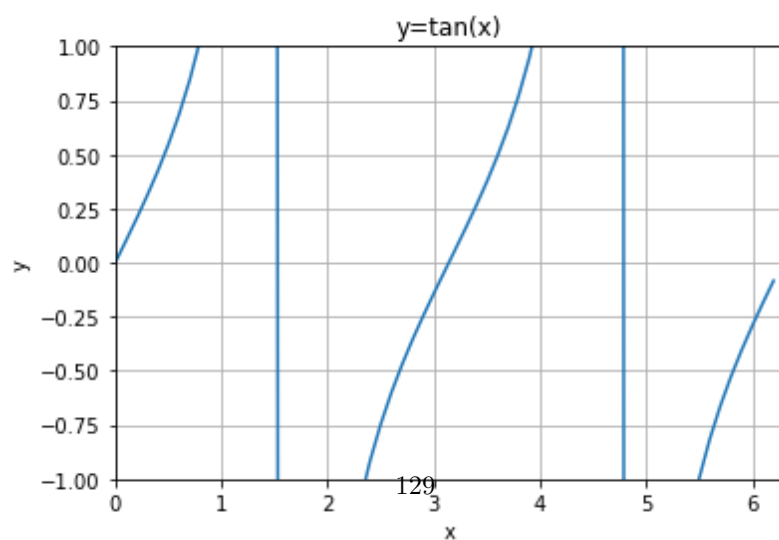
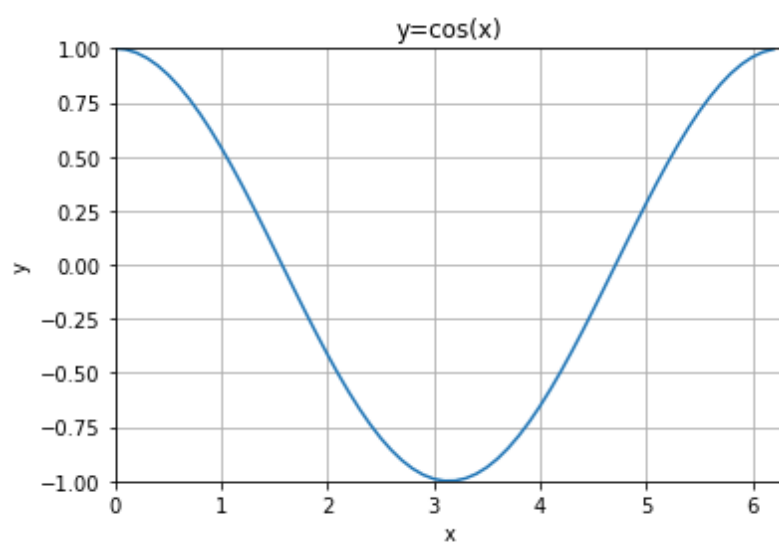
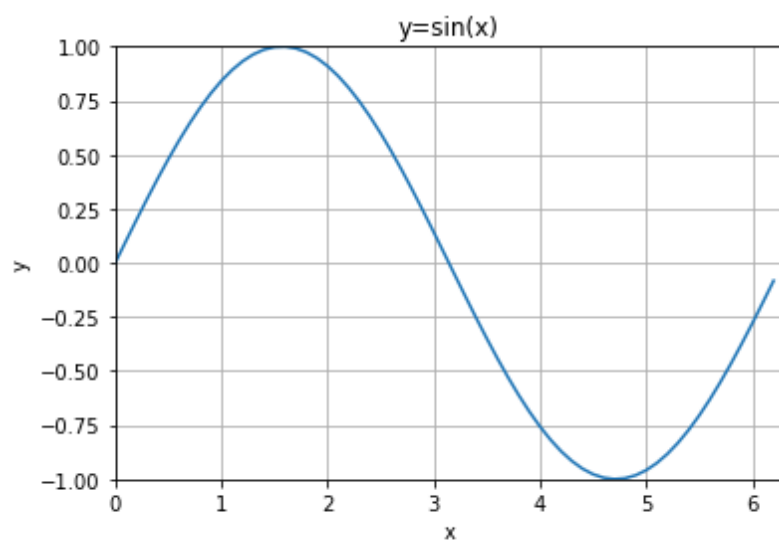


Figure 23.3: Trigonometric Functions

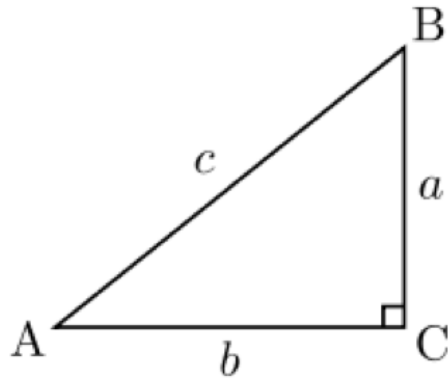


Figure 23.4: Right Triangle

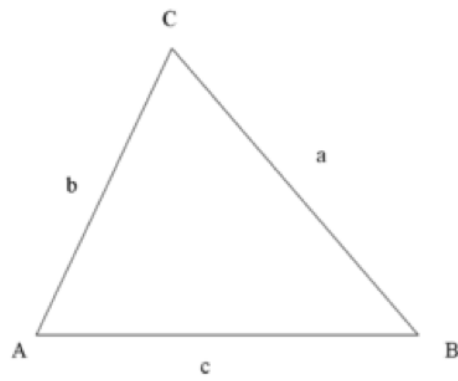


Figure 23.5: Law of Cosines

Chapter 24

Linear Algebra in Python

24.1 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:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \in R^{n \times m} \quad (24.1)$$

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

Example of a 3 x 3 matrix:

$$A = \begin{bmatrix} 1 & 5 & 3 \\ 4 & 6 & 6 \\ 3 & 8 & 9 \end{bmatrix} \quad (24.2)$$

Example of a 3 x 4 matrix:

$$A = \begin{bmatrix} 1 & 5 & 3 & 4 \\ 4 & 5 & 7 & 8 \\ 7 & 8 & 9 & 3 \end{bmatrix} \quad (24.3)$$

Example of a 4 x 2 matrix:

$$A = \begin{bmatrix} 1 & 5 \\ 4 & 5 \\ 3 & 2 \\ 7 & 8 \end{bmatrix} \quad (24.4)$$

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

Example 24.1.1. 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 24.1: 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]

24.2 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 24.2.1. 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
14 print("A =", A)
15
16
17
18
19 A = np.array([[0, 1],
20              [-2, -3]])
21
22 B = np.array([[1, 0],
23              [3, -2]])
24
25 C = A + B
26 print(C)
27
28
29 C = A.dot(B)
30 print(C)
31
32 C = A.transpose()
33 print(C)
```

Listing 24.2: Matrix manipulation using NumPy

[End of Example]

24.2.1 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 24.3: Matrix manipulation using NumPy

24.2.2 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)
```

24.2.3 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.

24.2.4 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:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \quad (24.5)$$

$$B = \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} \quad (24.6)$$

Then we get:

$$A+B = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} = \begin{bmatrix} 0+1 & 1+0 \\ -2+3 & -3-2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -5 \end{bmatrix} \quad (24.7)$$

Example 24.2.2. 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 24.4: Matrix Addition in Python

We get:

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

[End of Example]

24.2.5 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:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \quad (24.8)$$

$$B = \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} \quad (24.9)$$

Then we get:

$$A - B = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} = \begin{bmatrix} 0-1 & 1-0 \\ -2-3 & -3-(-2) \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ -5 & -1 \end{bmatrix} \quad (24.10)$$

Example 24.2.3. 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 24.5: Matrix Subtraction in Python

We get:

```

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

[End of Example]

24.2.6 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:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \quad (24.11)$$

$$B = \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} \quad (24.12)$$

Then we get:

$$\begin{aligned} AB &= \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} \\ &= \begin{bmatrix} 0 \cdot 1 + 1 \cdot 3 & 0 \cdot 0 + 1 \cdot (-2) \\ -2 \cdot 1 + (-3) \cdot 3 & -2 \cdot 0 + (-3) \cdot (-2) \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ -11 & 6 \end{bmatrix} \quad (24.13) \end{aligned}$$

We do the same in Python:

Example 24.2.4. 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 24.6: 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 24.7: 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 24.1.

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

Figure 24.1: Matrix Multiplication

We have also the following matrix rules:

$$AB \neq BA \quad (24.14)$$

$$A(BC) = (AB)C \quad (24.15)$$

$$(A + B)C = AC + BC \quad (24.16)$$

$$C(A + B) = CA + CB \quad (24.17)$$

Exercise 24.2.1. Matrix Rules

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

[End of Exercise]

24.2.7 Transpose of a Matrix

A general matrix is defined as:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \in R^{n \times m} \quad (24.18)$$

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

The transpose of matrix a is then:

$$A^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ a_{12} & a_{22} & \dots & a_{n2} \\ \dots & \dots & \dots & \dots \\ a_{1m} & a_{n2} & \dots & a_{nm} \end{bmatrix} \in R^{m \times n} \quad (24.19)$$

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

Example 24.2.5. 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
18 C = np.matrix([[1, 4],
19               [2, -3],
20               [-6, -2]])
21
22 Ct = np.transpose(C)
23 print (Ct)
```

Listing 24.8: 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]

24.2.8 Determinant

Given a matrix A the Determinant is given by:

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

For a 2x2 matrix A:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad (24.20)$$

We have:

$$\det(A) = |A| = a_{11}a_{22} - a_{21}a_{12} \quad (24.21)$$

Example:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad (24.22)$$

We get:

$$\det(A) = |A| = 1 \cdot 4 - 3 \cdot 2 = 4 - 6 = -2 \quad (24.23)$$

Example 24.2.6. 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 24.9: Determinant

This gives:

```
1 -2.0000000000000004
```

Listing 24.10: Determinant

[End of Example]

24.2.9 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:

$$AA^{-1} = A^{-1}A = I \quad (24.24)$$

For a 2x2 matrix we have:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad (24.25)$$

The inverse of A becomes:

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \quad (24.26)$$

Example 24.2.7. 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 24.11: Inverse Matrix

We get the following results:

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

[End of Example]

24.3 Solving Linear Equations

Example 24.3.1. Solving Linear Equations

Given the equations:

$$x_1 + 2x_2 = 5 \quad (24.27)$$

$$3x_1 + 4x_2 = 6 \quad (24.28)$$

We want to set the equations on the following form:

$$Ax = b \quad (24.29)$$

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:

$$x = A^{-1}b \quad (24.30)$$

We get:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad (24.31)$$

$$b = \begin{bmatrix} 5 \\ 6 \end{bmatrix} \quad (24.32)$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad (24.33)$$

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 24.3.2. 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
    exists
12
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]

24.4 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 24.4.1. Exercise Solving Linear Equations

Given the following equations:

$$x_1 + 2x_2 = 5 \quad (24.34)$$

$$3x_1 + 4x_2 = 6 \quad (24.35)$$

$$7x_1 + 8x_2 = 9 \quad (24.36)$$

Find the solutions for the given equations using Python.

[End of Exercise]

Exercise 24.4.2. 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 24.12: 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 25

Complex Numbers in Python

25.1 Introduction to Complex Numbers

A complex number is defined like this:

$$z = a + jb \quad (25.1)$$

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.:

$$\text{Re}(z) = a, \text{Im}(z) = b$$

Figure 25.1 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:

$$z^* = a - jb \quad (25.2)$$

You may also imagine numbers on exponential/polar form:

$$z = re^{j\theta} \quad (25.3)$$

Where:

$$r = |z| = \sqrt{a^2 + b^2} \quad (25.4)$$

and

$$\theta = \arctan \frac{b}{a} \quad (25.5)$$

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

Figure 25.2 shows an illustration of complex numbers on polar form.

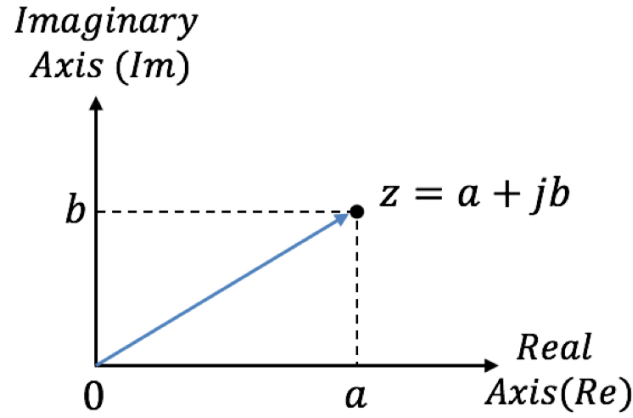


Figure 25.1: Complex Numbers

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:

$$z_1 = r_1 e^{j\theta_1} \quad (25.6)$$

$$z_2 = r_2 e^{j\theta_2} \quad (25.7)$$

Multiplication:

$$z_3 = z_1 z_2 = r_1 r_2 e^{j(\theta_1 + \theta_2)} \quad (25.8)$$

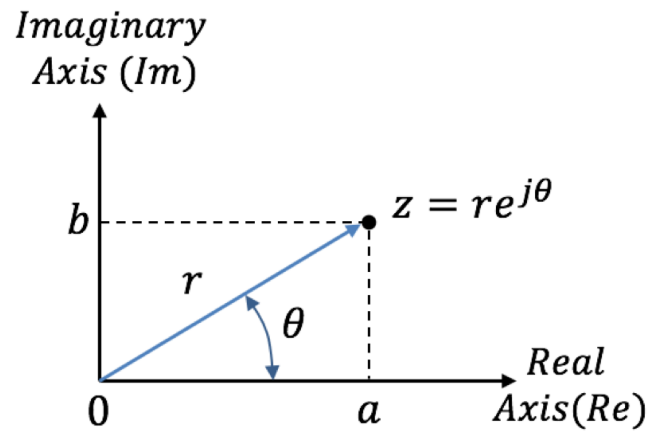


Figure 25.2: Complex Numbers - Polar form

Division:

$$z_3 = \frac{z_1}{z_2} = \frac{r_1 e^{j\theta_1}}{r_2 e^{j\theta_2}} = \frac{r_1}{r_2} e^{j(\theta_1 - \theta_2)} \quad (25.9)$$

25.2 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 25.2.1. Basic Complex Numbers in Python

Given the following complex numbers:

$$a = 5 + 3j \quad (25.10)$$

$$b = 1 - 1j \quad (25.11)$$

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 25.1: Basic Complex Numbers in Python

[End of Example]

Example 25.2.2. 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
18 # converting complex number into rectangular using rect()
19 w = cmath.rect(2,3)
20 print(w)

```

Listing 25.2: Complex Number Functions in Python

[End of Example]

Chapter 26

Differential Equations

26.1 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:

$$\frac{dy}{dt} = f(t, y), y(t_0) = y_0 \quad (26.1)$$

Note! Different notation is used: $\frac{dy}{dt} = y' = \dot{y}$

This document will use these different notations interchangeably.

Example 26.1.1. Example of Dynamic System

Given the following differential equation:

$$\dot{x} = -ax + bu \quad (26.2)$$

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 35 we will simulate (solve and plot the results) such differential equations numerically using Euler discretization.

Example 26.1.2. Differential Equation Example

Given the following differential equation:

$$\dot{x} = ax \quad (26.3)$$

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:

$$x(t) = e^{at}x_0 \quad (26.4)$$

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 26.1: Differential Equation

This gives the plot shown in Figure 26.1.

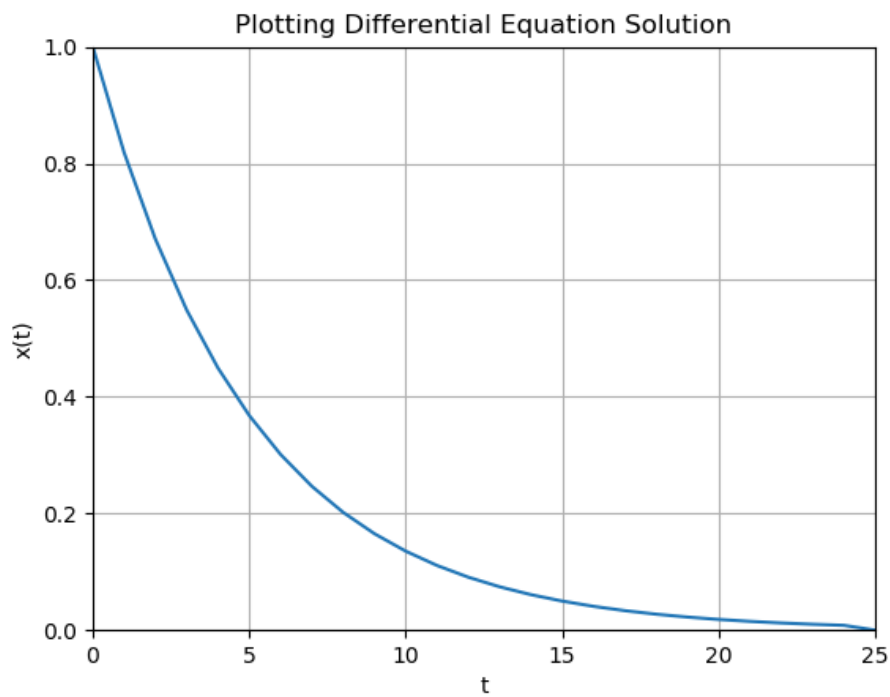


Figure 26.1: Plotting Differential Equation Solution

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 26.2: Differential Equation

This alternative Python code gives the same plot as shown in Figure 26.1.

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 35).

[End of Example]

26.2 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 26.2.1. Using ODE Solver in Python

Given the following differential equation:

$$\dot{x} = ax \quad (26.5)$$

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 = 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
21     return dxdt
22
23
24 # Solve ODE
25 x = odeint(mydiff, x0, t)
26 print(x)
27
28
29 # Plot the Results
30 plt.plot(t,x)
31 plt.title('Plotting Differential Equation Solution')
32 plt.xlabel('t')
33 plt.ylabel('x(t)')
34 plt.grid()
35 plt.axis([0, 25, 0, 1])
36 plt.show()

```

Listing 26.3: Using ODE Solver in Python

This gives the same plot as shown in Figure 26.1.

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 26.4: Using ODE Solver in Python

This gives the same plot as shown in Figure 26.1.

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]

26.3 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 26.3.1. Set of 1.order Differential Equations

Given the differential equations:

$$\frac{dx}{dt} = -y \quad (26.6)$$

$$\frac{dy}{dt} = x \quad (26.7)$$

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+1, increment)
15
16
17 # Function that returns dx/dt
18 def mydiff(z, t):
19     dxdt = -z[1]
20     dydt = z[0]
21
22     dzdt = [dxdt, dydt]
23     return dzdt
24
25
26 # Solve ODE
27 z = odeint(mydiff, z0, t)
28 print(z)
29
30 x = z[:,0]
31 y = z[:,1]
32
33
34 # Plot the Results
35 plt.plot(t,x)
36 plt.plot(t,y)
37 plt.title('Plotting Differential Equations Solution')
38 plt.xlabel('t')
39 plt.ylabel('z(t)')
40 plt.grid()
41 plt.axis([-1, 1, -1.5, 1.5])
42 plt.show()

```

Listing 26.5: xxx

This gives the the plot shown in Figure 26.2.

We can also rewrite the differential equations like this (to make it easier to understand?):

$$\frac{dx_1}{dt} = -x_2 \quad (26.8)$$

$$\frac{dx_2}{dt} = x_1 \quad (26.9)$$

The Python code then becomes:

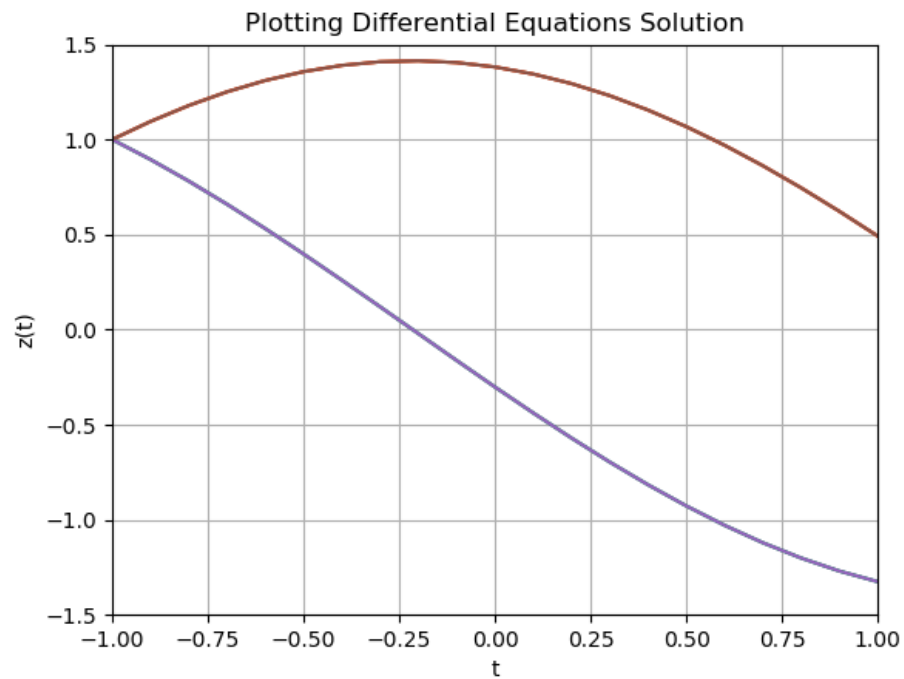


Figure 26.2: Figure Name

```

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+1, increment)
15
16
17 # Function that returns dx/dt
18 def mydiff(x, t):
19     dx1dt = -x[1]
20     dx2dt = x[0]
21
22     dxdt = [dx1dt, dx2dt]
23     return dxdt
24
25
26 # Solve ODE
27 x = odeint(mydiff, x_init, t)
28 print(x)
29
30 x1 = x[:,0]

```

```

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 26.6: xxx

The plot and results will be the same.

[End of Example]

26.4 Solving Higher order Differential Equations

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

$$(1 + t^2)\ddot{w} + 2t\dot{w} + 3w = 2 \quad (26.10)$$

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:

$$\ddot{w} = \frac{2 - 2t\dot{w} - 3w}{1 + t^2} \quad (26.11)$$

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

We do the following "trick":

$$w = x_1 \quad (26.12)$$

$$\dot{w} = x_2 \quad (26.13)$$

This gives a set of 1.order differential equations:

$$\dot{x}_1 = x_2 \quad (26.14)$$

$$\dot{x}_2 = \frac{2 - 2tx_2 - 3x_1}{1 + t^2} \quad (26.15)$$

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+1, 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 26.7: xxx

This gives the the plot shown in Figure 26.3.

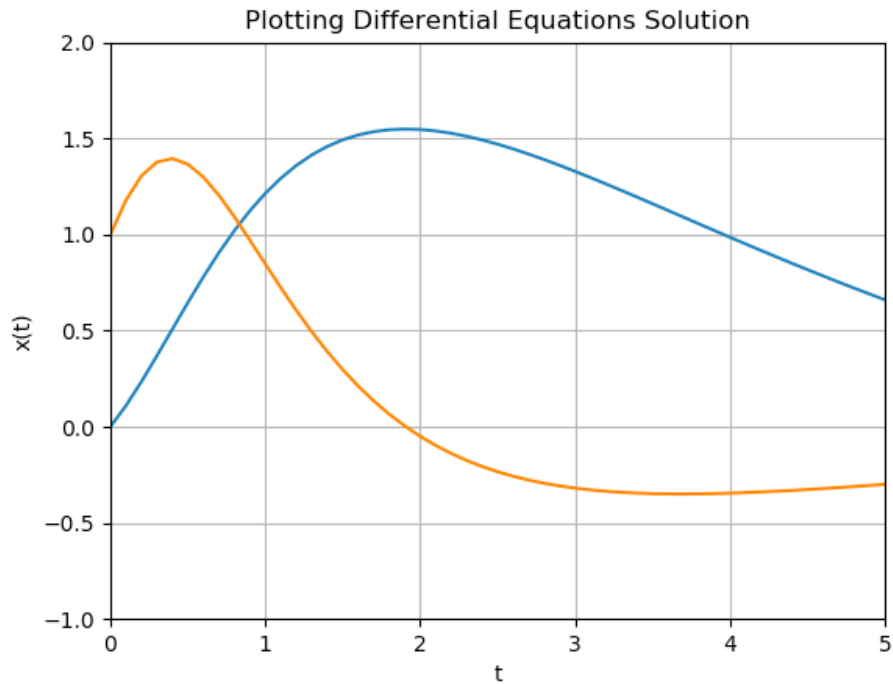


Figure 26.3: 2.order Differential Equation

26.5 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 26.5.1. Bacteria Population

In this task we will simulate a simple model of a bacteria population in a jar.

The model is as follows:

$$\text{birth rate} = bx \quad (26.16)$$

$$\text{death rate} = px^2 \quad (26.17)$$

Then the total rate of change of bacteria population is:

$$\dot{x} = bx - px^2 \quad (26.18)$$

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 26.5.2. Differential Equation

Given the following differential equation:

$$\dot{x} = ax + b \quad (26.19)$$

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 26.5.3. Simulation of Dynamic System

Given the following differential equation:

$$\dot{x} = -ax + bu \quad (26.20)$$

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 26.5.4. Mass-Spring-Damper System

Given a "Mass-Spring-Damper" system as shown in Figure 26.4.

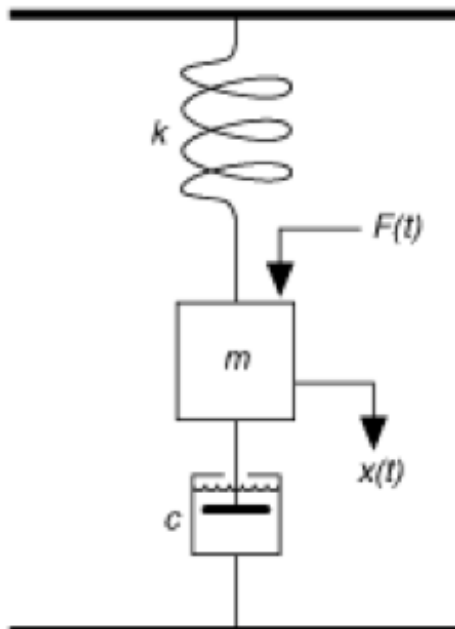


Figure 26.4: Mass-Spring-Damper System

The system can be described by the following equation:

$$F(t) - c\dot{x}(t) - kx(t) = m\ddot{x}(t) \quad (26.21)$$

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 26.5.5. ODE

Use the one of the ODE solvers in Python to solve and plot the results of the following differential equation in the interval $[t_0, t_f]$:

$$3w' + \frac{1}{1+t^2}w = \cos(t) \quad (26.22)$$

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 26.5.6. Pendulum model

Use the one of the ODE solvers in Python to solve and plot the results of the following differential equations:

$$\dot{x}_1 = x_2 \quad (26.23)$$

$$\dot{x}_2 = -\frac{g}{r}x_1 - \frac{b}{mr^2}x_2 \quad (26.24)$$

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.

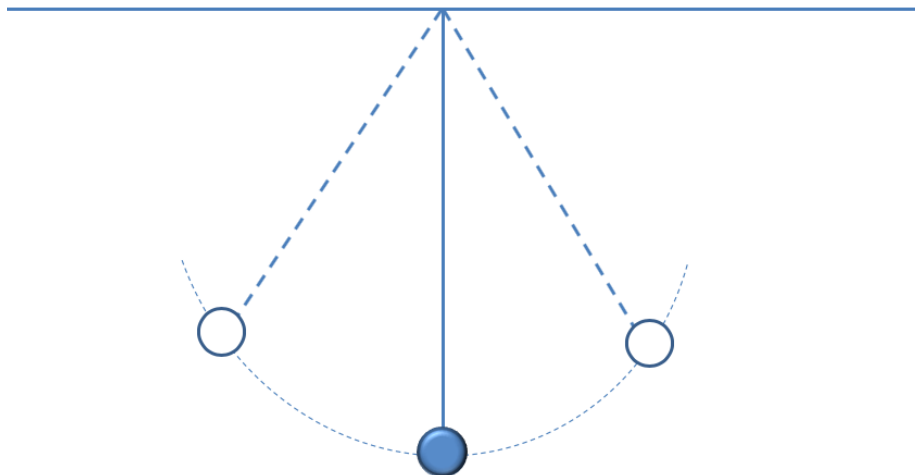


Figure 26.5: Pendulum

A pendulum is shown in Figure 26.5.

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 27

Interpolation

Interpolation is used to estimate data points between two known points. The most common interpolation technique is Linear Interpolation.

We can use the following packages:

numpy.interp

<https://docs.scipy.org/doc/numpy/reference/generated/numpy.interp.html>

scipy.interpolate

<https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html>

Example 27.0.1. Basic Interpolation Example

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x = [1, 2, 3]
5 y = [3, 2, 0]
6
7 x_new = 2.5
8
9 y_new = np.interp(2.5, x, y)
10
11 print("New Interpolated Value:")
12 print(y_new)
13
14 plt.plot(x, y, 'o-')
15 plt.show()
```

Listing 27.1: Basic Interpolation Example

Figure 27.1 shows the plot given by the code.

By using the interpolation function **interp** that is part of the **numpy** library we can find an interpolated value for $x = 2.5$ in this example. Python gives the following results:

```
1 New Interpolated Value:
2 1.0
```

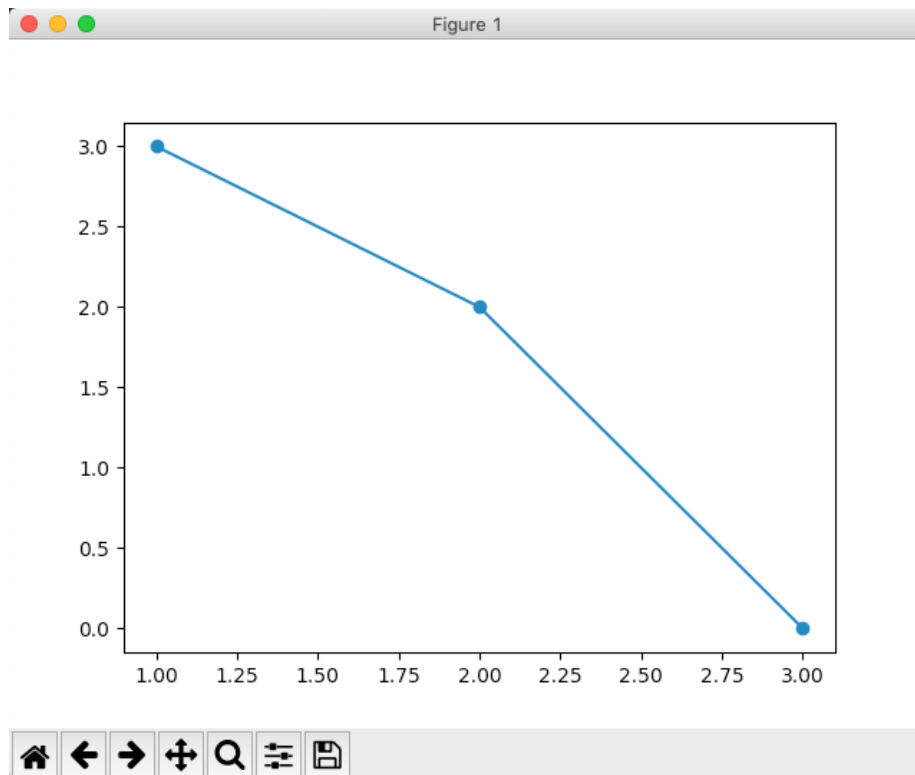


Figure 27.1: Basic Interpolation Example

And seen from Figure 27.1 that is the expected result.

[End of Example]

Example 27.0.2. Interpolation Example using `scipy.interpolate`

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy import interpolate
4
5 x = np.arange(0, 10)
6 y = np.exp(-x/3.0)
7 f = interpolate.interpld(x, y)
8
9 xnew = np.arange(0, 9, 0.1)
10 ynew = f(xnew) # use interpolation function returned by 'interpld'
11 plt.plot(x, y, 'o', xnew, ynew, '-')
12 plt.show()

```

Listing 27.2: Interpolation Example

Figure 27.2 shows the plot generated by the code.

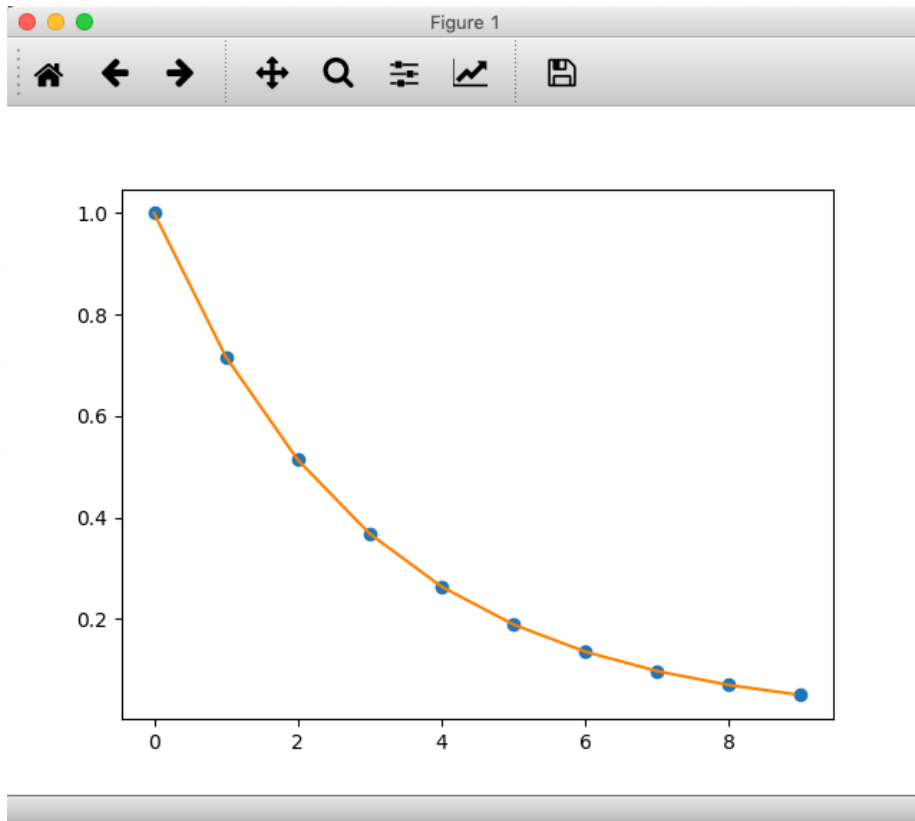


Figure 27.2: Interpolation Example

[End of Example]

Example 27.0.3. Another Interpolation Example using `scipy.interpolate`

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.interpolate import interp1d
4
5 x = np.linspace(0, 10, num=11, endpoint=True)
6 y = np.cos(-x**2/9.0)
7
8 f = interp1d(x, y) #linear is default
9 f2 = interp1d(x, y, kind='cubic')
10
11
12 xnew = np.linspace(0, 10, num=41, endpoint=True)
13
14 plt.plot(x, y, 'o', xnew, f(xnew), '-', xnew, f2(xnew), '-')
15 plt.legend(['data', 'linear', 'cubic'], loc='best')
16 plt.show()

```

Listing 27.3: Interpolation Example

Figure 27.3 shows the plot generated by the code.

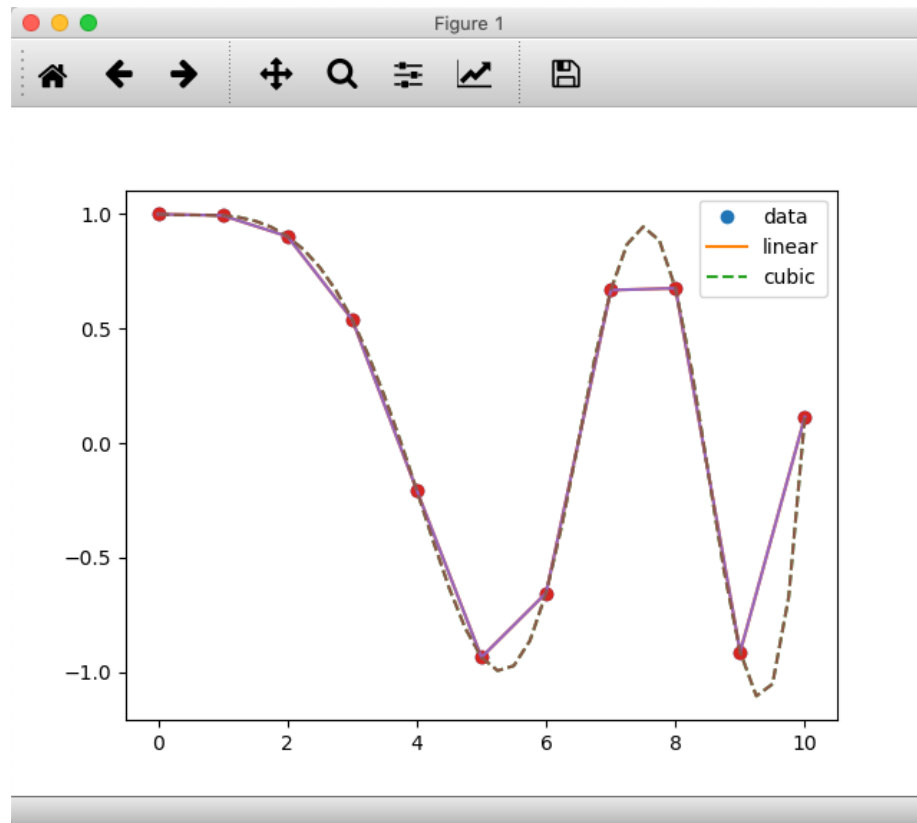


Figure 27.3: Interpolation Example

From Figure 27.3 we see that the cubic interpolation gives a good results based on the cosine function we used in this example.

[End of Example]

27.1 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 27.1.1. Basic Interpolation

Given the following data as shown in Table 28.2.
Find the interpolated value for $x = 3.5$.

You should try with different types of interpolation.

You should also plot the data points and the interpolated data in the same plot.

[End of Exercise]

Exercise 27.1.2. Interpolation

Given the following data as shown in Table 28.2.

Plot u versus T . Find the interpolated data and plot it in the same graph.

Test out different interpolation types.

Discuss the results. What kind of interpolation is best in this case?

What is the interpolated value for $u=2680.78$ KJ/kg?

[End of Exercise]

Table 27.1: Data	
x	y
0	15
1	10
2	9
3	6
4	2
5	0

Table 27.2: Data	
Temperature T [C]	Energy u [KJ/kg]
100	2506.7
150	2582.8
200	2658.1
250	2733.7
300	2810.4
400	2967.9
500	3131.6

Chapter 28

Curve Fitting - Fitting Models to Data

In the previous chapter we found interpolated points, i.e., we found values between the measured points using the interpolation technique. It would be more convenient to model the data as mathematical function $y=f(x)$. Then we could easily calculate any data we want based on this model.

We will use Python to create empiric data model. It is important to have in mind that these models are good only in the region we have collected data.

These techniques use a polynomial of degree N that fits the data Y best in a least-squares sense. A polynomial is expressed as:

$$p(x) = p_1x^n + p_2x^{n-1} + \dots + p_nx + p_{n+1} \quad (28.1)$$

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

We have Linear Regression and Polynomial Regression.

We can use the `curve_fit` function in the Scipy Python library to help us with curve fitting.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html

28.1 Linear Regression

Here we will create a linear model of our data on the form:

$$y = ax + b \quad (28.2)$$

This is actually a polynomial of 1. order.

Example 28.1.1. Curve Fitting

Given the following data as shown in Table 28.1.

We want to determine the slope (a) and intercept (b) of the best-fit line to the data.

Python Code:

```
1 from scipy.optimize import curve_fit
2
3 x = [0, 1, 2, 3, 4, 5]
4 y = [15, 10, 9, 6, 2, 0]
5
6
7 def linear_model(x, a, b):
8     return a * x + b
9
10 popt, pcov = curve_fit(linear_model, x, y)
11
12 print(popt)
```

Listing 28.1: Curve Fitting Example

The `curve_fit` function returns two items, which we can `popt` and `pcov`. The `popt` argument are the best-fit parameters for `a` and `b`.

The `pcov` variable contains the covariance matrix, which indicates the uncertainties and correlations between parameters.

From the Python code we get the following results:

```
[-2.91428571 14.28571429]
```

This means $a = -2.91428571$ and $b = 14.28571429$.

Based on that we get the following model based on the given data:

$$y = -2.9x + 14.3 \quad (28.3)$$

Next, it is also a good idea to plot the actual data in the same plot as the model for comparison.

We extend the code as follows:

```
1 import numpy as np
2 from scipy.optimize import curve_fit
3 import matplotlib.pyplot as plt
4
5 x = [0, 1, 2, 3, 4, 5]
```



```

6 y = [15, 10, 9, 6, 2, 0]
7
8
9 def linear_model(x, a, b):
10     return a * x + b
11
12 popt, pcov = curve_fit(linear_model, x, y)
13
14 print(popt)
15
16 plt.plot(x,y, 'or')
17
18
19 xstart = -1
20 xstop = 6
21 increment = 0.1
22 xmodel = np.arange(xstart, xstop, increment)
23
24 a = popt[0]
25 b = popt[1]
26
27 ymodel = a*xmodel + b
28
29 plt.plot(xmodel, ymodel, 'b')

```

Listing 28.2: Curve Fitting Example

Figure 28.1 shows the plot generated by the code.

[End of Example]

28.2 Polynomial Regression

In the previous section we used linear regression which is a 1. order polynomial. In this section we will study higher order polynomials. In polynomial regression we will find the following model:

$$y(x) = a_0x^n + a_1x^{n-1} + \dots + a_{n-1}x + a_n \quad (28.4)$$

Example 28.2.1. Curve Fitting with Polynomial Regression

Given the following data as shown in Table 28.1.

We want to find models on the form:

$$y(x) = a_0x^n + a_1x^{n-1} + \dots + a_{n-1}x + a_n \quad (28.5)$$

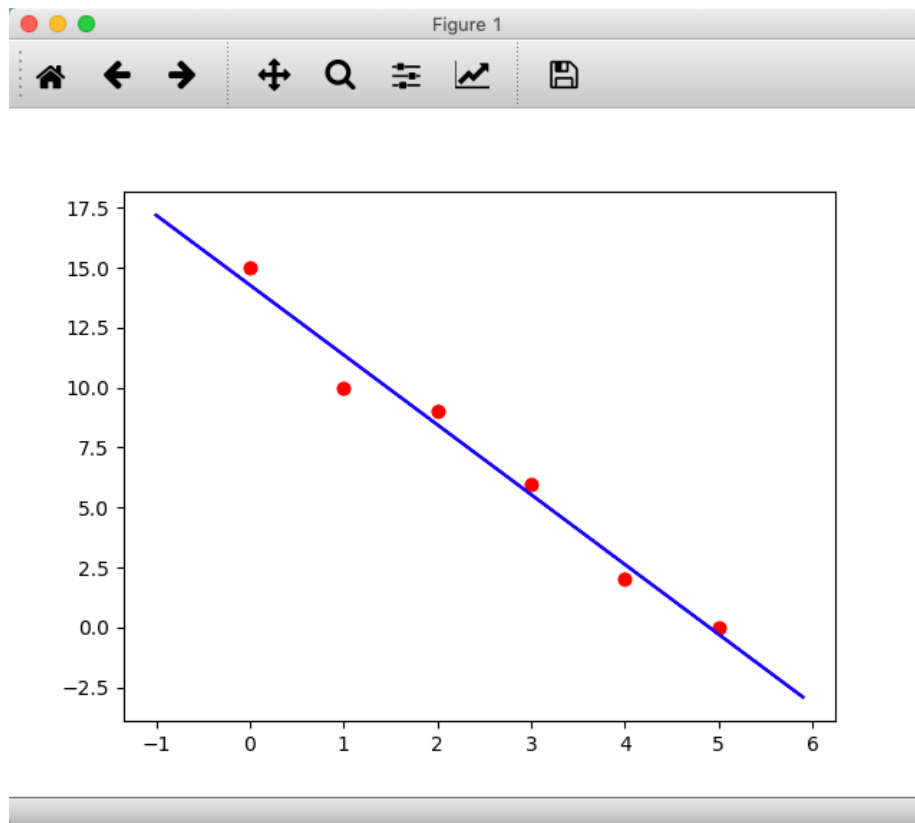


Figure 28.1: Interpolation Example

We will use the Python to find and compare the models using different orders of the polynomial.

We will investigate models of 2.order, 3.order, 4.order and 5.order.

We have only 6 data points, so a model with order higher than 5 will make no sense.

We start with a 2.order model:

$$y(x) = ax^2 + bx + c \quad (28.6)$$

Python Code:

```
1 import numpy as np
2 from scipy.optimize import curve_fit
3 import matplotlib.pyplot as plt
4
```

```

5 x = [0, 1, 2, 3, 4, 5]
6 y = [15, 10, 9, 6, 2, 0]
7
8
9 def linear_model(x, a, b, c):
10     return a * x**2 + b * x + c
11
12 popt, pcov = curve_fit(linear_model, x, y)
13
14 print(popt)
15
16 plt.plot(x,y, 'ok')
17
18
19 xstart = -1
20 xstop = 6
21 increment = 0.1
22 xmodel = np.arange(xstart, xstop, increment)
23
24 a = popt[0]
25 b = popt[1]
26 c = popt[2]
27
28 ymodel = a * xmodel**2 + b * xmodel + c
29
30 plt.plot(xmodel, ymodel, 'b')

```

Listing 28.3: Curve Fitting Example - 2.order model

The Python code gives the following results:

```
[0.05357143 -3.18214286 14.46428571]
```

This is the coefficients a, b and c in the model.

We can change the Python code to make a 3.order, 4.order and 5.order as well.

Here is the Python code for a 5.order model (based on that you can easily create code for 3.order and 4.order models):

```

1 import numpy as np
2 from scipy.optimize import curve_fit
3 import matplotlib.pyplot as plt
4
5 x = [0, 1, 2, 3, 4, 5]
6 y = [15, 10, 9, 6, 2, 0]
7
8
9 def linear_model(x, a, b, c, d, e, f):
10     return a * x**5 + b * x**4 + c * x**3 + d * x**2 + e * x + f
11
12 popt, pcov = curve_fit(linear_model, x, y)
13
14 print(popt)
15
16 plt.plot(x,y, 'ok')
17
18

```

```

19 xstart = -1
20 xstop = 6
21 increment = 0.1
22 xmodel = np.arange(xstart, xstop, increment)
23
24 a = popt[0]
25 b = popt[1]
26 c = popt[2]
27 d = popt[3]
28 e = popt[4]
29 f = popt[5]
30
31 ymodel = a * xmodel**5 + b * xmodel**4 + c * xmodel**3 + d * xmodel
    **2 + e * xmodel + f
32
33 plt.plot(xmodel, ymodel, 'm')

```

Listing 28.4: Curve Fitting Example - 5.order model

Figure 28.2 shows the plot generated by the code for the different models.

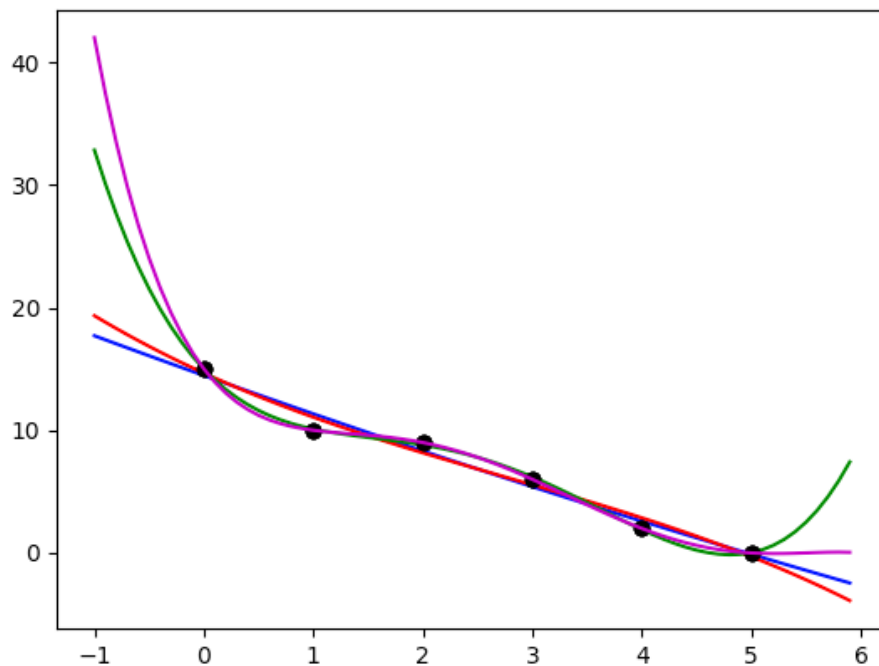


Figure 28.2: Interpolation Example

As expected, the higher order models match the data better and better.

Note! The fifth order model matches exactly because there were only six data points available.

It is important to have in mind that these models are good only in the region

we have collected data.

[End of Example]

Example 28.2.2. Polynomial Regression using polyfit and polyval functions

In this example we will use the NumPy functions **polyfit** and **polyval**.

We start with a 3.order model:

$$y(x) = ax^3 + bx^2 + cx + d \quad (28.7)$$

Python code:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Original Data
5 x = [0, 1, 2, 3, 4, 5]
6 y = [15, 10, 9, 6, 2, 0]
7
8 plt.plot(x,y, 'ok')
9
10
11 # Finding the Model
12 model_order = 3
13
14 p = np.polyfit(x, y, model_order)
15 print(p)
16
17 # Plot the Model
18 xstart = -1
19 xstop = 6
20 increment = 0.1
21 xmodel = np.arange(xstart, xstop, increment)
22
23 ymodel = np.polyval(p, xmodel)
24
25 plt.plot(xmodel, ymodel, 'r')
```

Listing 28.5: Polynomial Regression using polyfit and polyval functions

We get the following results:

[-0.06481481 0.53968254 -4.07010582 14.65873016]

This means the following 3.order model:

$$y(x) = -0.06x^3 + 0.54x^2 - 4.1x + 14.7 \quad (28.8)$$

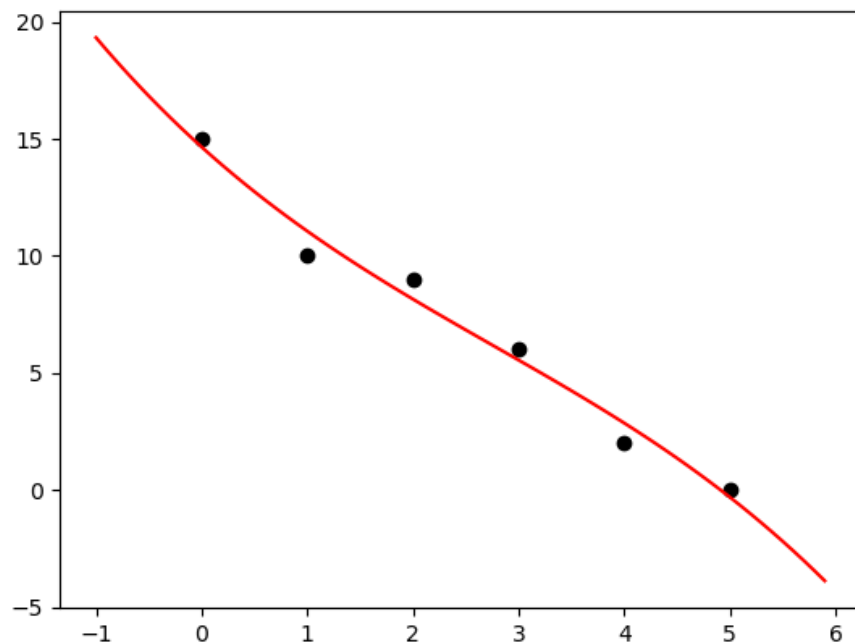


Figure 28.3: Interpolation Example - 3.order model

Figure 28.3 shows the plot generated by the code.

Lets extend the code by creating different models with different orders. For easy comparison of different models in the same program we can use a For loop as shown in the code example below.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Original Data
5 x = [0, 1, 2, 3, 4, 5]
6 y = [15, 10, 9, 6, 2, 0]
7
8 plt.plot(x,y, 'ok')
9
10 # x values for model
11 xstart = -1
12 xstop = 6
13 increment = 0.1
14 xmodel = np.arange(xstart, xstop, increment)
15
16
17 startorder = 1
18 endorder = 5
19
20 for model_order in range(startorder, endorder, 1):
21
22     # Finding the Model
23     p = np.polyfit(x, y, model_order)

```

```

24     print(p)
25
26
27     # Plot the Model
28     ymodel = np.polyval(p, xmodel)
29
30     plt.plot(xmodel, ymodel)

```

Listing 28.6: Polynomial Regression using polyfit and polyval functions

The results are the same as shown in Figure 28.2.

[End of Example]

28.3 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 28.3.1. Interpolation

Given the following data as shown in Table 28.2.
Start by plotting u versus T .

Next you should create a linear model of our data on the form:

$$y = ax + b \quad (28.9)$$

Plot the data and the model in the same plot for comparison.

Test also other interpolation types.

Discuss the results. What kind of interpolation is best in this case?

[End of Exercise]

Exercise 28.3.2. Polynomial Interpolation

Given the following data as shown in Table 28.3.
Use the polyfit and polyval functions in Python and compare the models using different orders of the polynomial.

Use subplots and make sure to add titles, etc.

[End of Exercise]

Table 28.1: Data

x	y
0	15
1	10
2	9
3	6
4	2
5	0

Table 28.2: Data

Temperature T [C]	Energy u [KJ/kg]
100	2506.7
150	2582.8
200	2658.1
250	2733.7
300	2810.4
400	2967.9
500	3131.6

Table 28.3: Example Data

x	y
10	23
20	45
30	60
40	82
50	111
60	140
70	167
80	198
90	200
100	220

Table 28.4: Example Data

Height, h [ft]	Flow, f [ft ³ /s]
0	0
1.7	2.6
1.9 5	3.6
2.60	4.03
2.92	6.45
4.04	11.22
5.24	30.61

Exercise 28.3.3. Model Fitting

Given the following data as shown in Table 28.4.

Create a 1. (linear), 2. (quadratic) and 3.order (cubic) model. Which gives the best model?

Plot the result in the same plot and compare them. Add xlabel, ylabel, title and a legend to the plot and use different line styles so the user can easily see the difference.

[End of Exercise]

Chapter 29

Least Square Method

The least squares method requires the model to be set up in the following form based on input-output data:

$$Y = \Phi\theta \quad (29.1)$$

Where:

θ is a vector with the unknown parameters on which we want to find the values.

Y is a vector with known measurements.

Φ is the so-called regression matrix. This matrix consists of known values.

From mathematics we have the corresponding $b = Ax$ where we want to solve a linear equation system. In principle it is the same, but as you can see we use a slightly different notation in system identification.

Assume we have some data and we want to find a mathematical model that fits the data.

We can illustrate it with a simple sketch, see Figure 29.1.

We want to minimize the sum of the deviations $e_1, e_2, e_3, \dots, e_m$

We can define this as the following criteria function:

$$V(\theta) = e_1^2 + e_2^2 + e_3^2 + \dots + e_m^2 \quad (29.2)$$

We find the minimum by setting the derivative equal to 0:

$$\frac{dV}{d\theta} = 0 \quad (29.3)$$

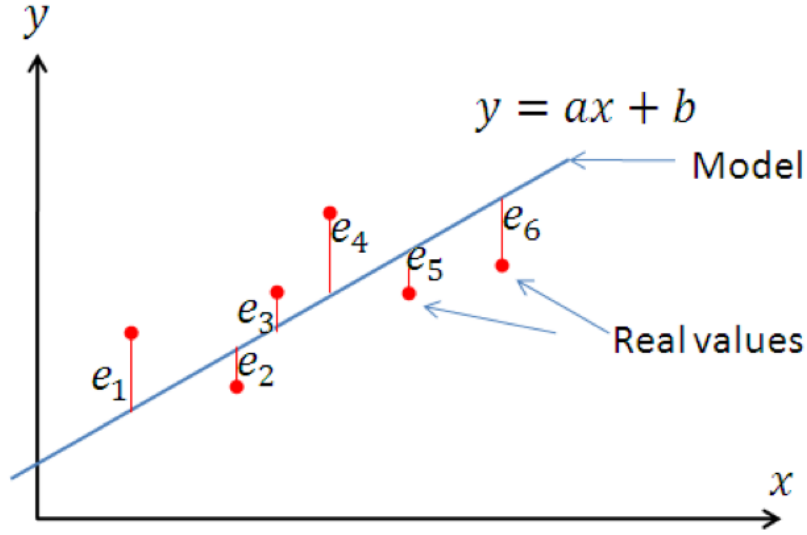


Figure 29.1: Least Square Principles

See Figure 29.2.

Based in this we can find the following Least Square formula:

$$\theta_{LS} = (\Phi^T \Phi)^{-1} \Phi^T Y \quad (29.4)$$

We will derive this equation later, but first we should go through some basic examples.

Example 29.0.1. Basic Least Square Method

Given the following model:

$$y(u) = au + b \quad (29.5)$$

Where the model parameters a and b are unknown.

The following values were found from experiments:

$$y(1) = 0.8 \quad (29.6)$$

$$y(2) = 3.0 \quad (29.7)$$

$$y(3) = 4.0 \quad (29.8)$$

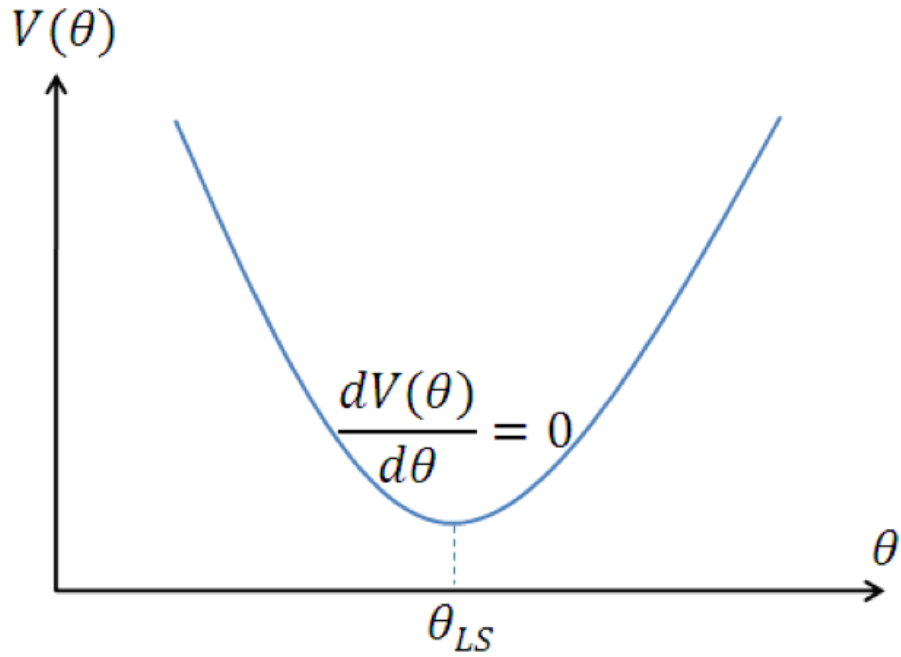


Figure 29.2: Finding Minimum of Criteria Function

We want to find the unknown model parameters a and b using the least squares method in Python.

First, we need to put it on Least Square form.

We have:

$$Y = \Phi\theta \quad (29.9)$$

Then we get:

$$0.8 = a1 + b \quad (29.10)$$

$$3.0 = a2 + b \quad (29.11)$$

$$0.8 = a1 + b \quad (29.12)$$

This becomes ($Y = \Phi\theta$):

$$\begin{bmatrix} 0.8 \\ 3.0 \\ 4.0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \quad (29.13)$$

We use the Least Square formula:

$$\theta_{LS} = (\Phi^T \Phi)^{-1} \Phi^T Y \quad (29.14)$$

Lets implement this formula with Python:

```
1 import numpy as np
2
3 Phi = np.array([[1, 1],
4                 [2, 1],
5                 [3, 1]])
6
7 Y = np.array([[0.8],
8               [3.0],
9               [4.0]])
10
11
12 #Method 1 - Ax = b (Not working in this case)
13 #theta = np.linalg.solve(Phi, Y) #Not working because inverse(Phi)
    does not exists
14
15
16 #Method 2 - using built-in Least Square Function in NumPy
17 theta = np.linalg.lstsq(Phi, Y, rcond=None)[0]
18
19 print(theta)
20
21
22 #Method 3 - Implmenting Least Square from scratch
23 theta = np.linalg.inv(Phi.transpose() * np.mat(Phi)) * Phi.
    transpose() * Y
24
25 print(theta)
```

Listing 29.1: xxx

The Python code gives the following:

```
1 [[ 1.6]
2  [-0.6]]
```

[End of Example]

Chapter 30

Numerical Differentiation

The derivative of a function $y = f(x)$ is a measure of how y changes with x .

Assume the following (Figure 30.1):

Then we have the following definition:

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (30.1)$$

In Python we typically deal with numerical calculations, so we will focus on finding a numerical approach and use different functions for numerical differentiation.

A numerical approach to the derivative of a function $y = f(x)$ is:

$$\frac{dy}{dx} = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \quad (30.2)$$

This approximation of the derivative corresponds to the slope of each line segment used to connect each data point that exists.

We can use the NumPy `diff()` function for implementation of this numerical approach. Several examples will be provided below.

Example 30.0.1. Numerical Differentiation

We will use numerical differentiation to find $\frac{dy}{dx}$ on the following function:

$$y = x^2 \quad (30.3)$$

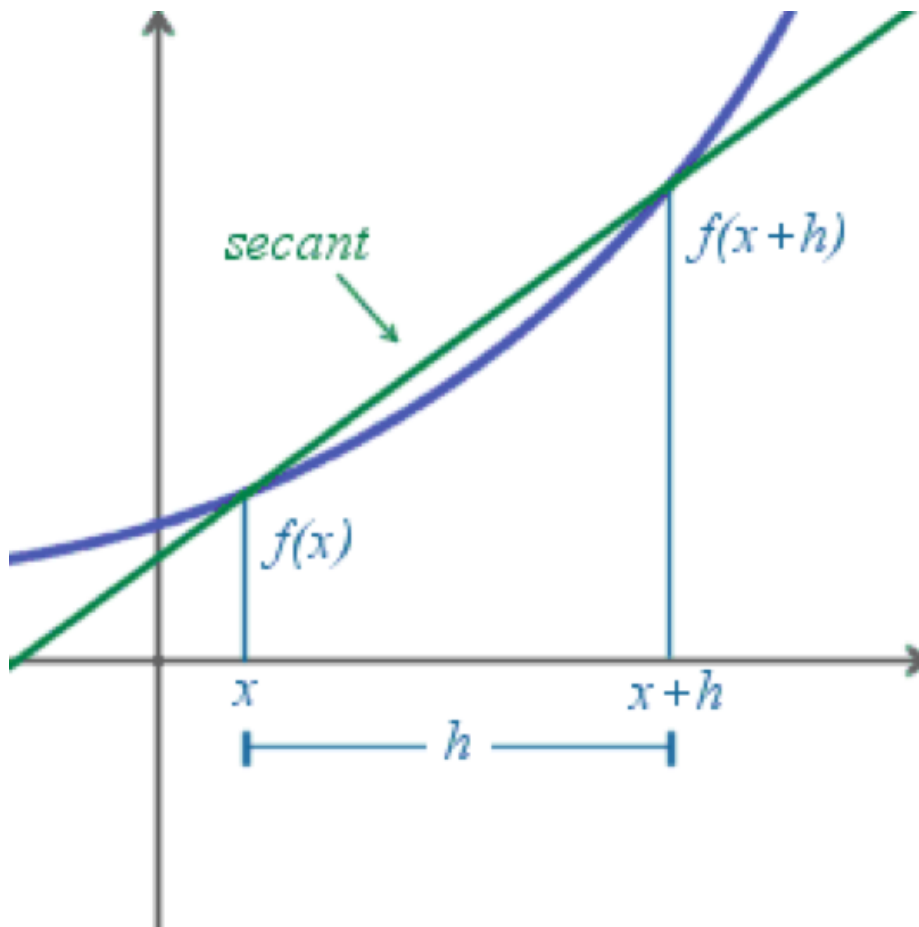


Figure 30.1: Numerical Differentiation

We use the following data as shown in Table 30.1.

Python code:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 xstart = -2
6 xstop = 2.1
7 increment = 0.1
8 x = np.arange(xstart, xstop, increment)
9
10 y = x**2
11
12 plt.plot(x, y)
13
14
15 xstart = -2
16 xstop = 3
17 increment = 1
18 x = np.arange(xstart, xstop, increment)

```

```

19
20 y = x**2;
21
22 plt.plot(x,y, 'ro')

```

Listing 30.1: Numerical Differentiation

This gives the the plot shown in Figure 30.2.

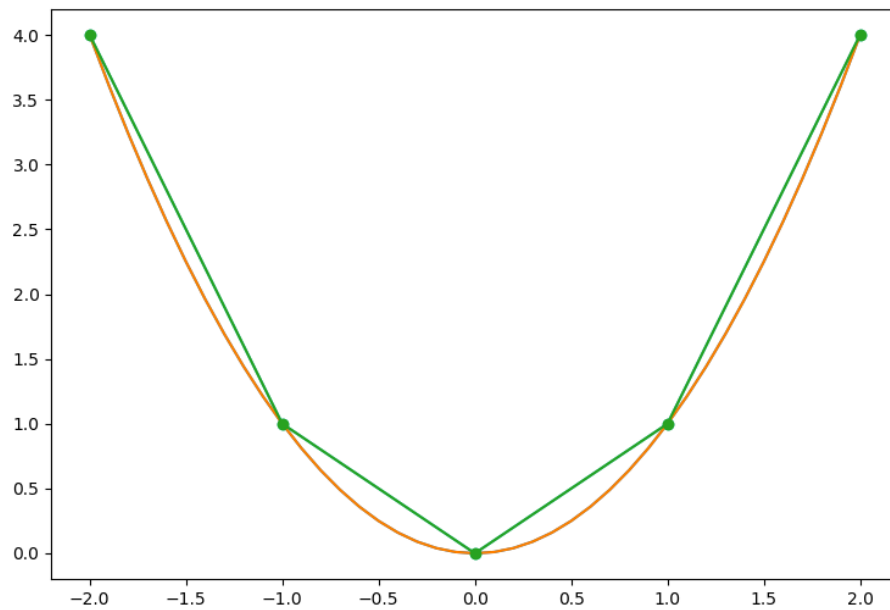


Figure 30.2: Basic Numerical Differentiation Example

We know that the exact solution to the problem is:

$$\frac{dy}{dx} = 2x \quad (30.4)$$

For the values given in Table 30.1 we have the following:

$$\frac{dy}{dx}(x = -2) = -4 \quad (30.5)$$

$$\frac{dy}{dx}(x = -1) = -2 \quad (30.6)$$

$$\frac{dy}{dx}(x = 0) = 0 \quad (30.7)$$

$$\frac{dy}{dx}(x = 1) = 2 \quad (30.8)$$

$$\frac{dy}{dx}(x=2) = 4 \quad (30.9)$$

In the next code snippet we will calculate the numerical values and compare with the exact solutions which are listed above.

Python code:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 xstart = -2
6 xstop = 3
7 increment = 1
8
9 x = np.arange(xstart, xstop, increment)
10
11 y = x**2;
12
13
14 # Exact/Analytical Solution
15
16 dydx_exact = 2*x
17
18 print(dydx_exact)
19
20 plt.plot(x, dydx_exact, 'o-')
21
22
23
24 # Exact/Analytical Solution
25
26 dydx_num = np.diff(y) / np.diff(x);
27
28 print(dydx_num)
29
30 xstart = -2
31 xstop = 2
32
33 x = np.arange(xstart, xstop, increment)
34
35
36 plt.plot(x, dydx_num, 'o-')
```

Listing 30.2: Numerical Differentiation

We get the following results:

```
dydx_exact = [-4 -2 0 2 4]
dydx_num = [-3. -1. 1. 3.]
```

This gives the the plot shown in Figure 30.3.

We see that there are some differences between the exact solutions and the numeric solutions.

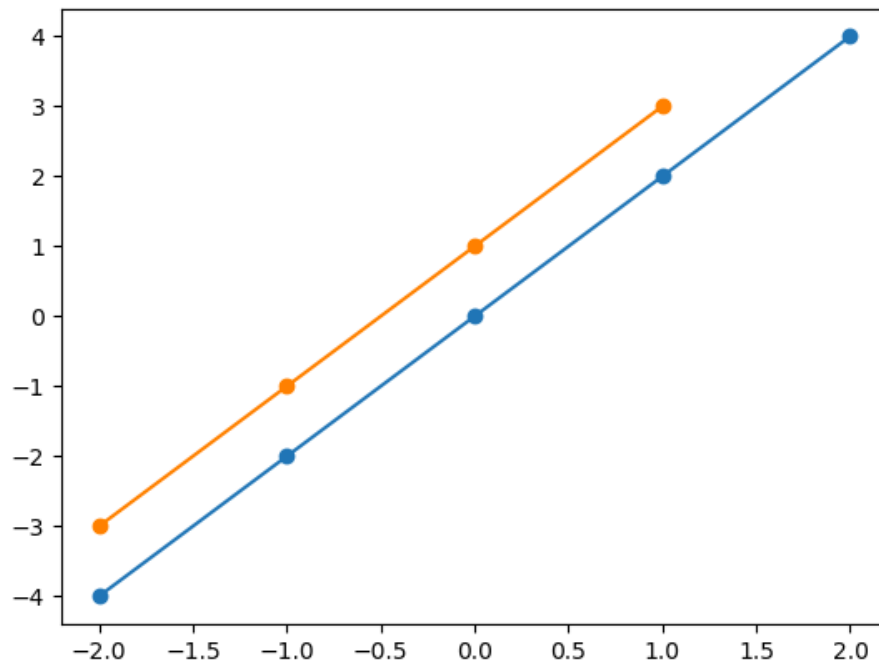


Figure 30.3: Numerical Differentiation Example

But what happens if we decrease the interval (h in Figure 30.1) between 2 points?

In the next code snippet we will decrease the interval between 2 points to $h = 0.1$.

Python code:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 xstart = -2
6 xstop = 2.1
7 increment = 0.1
8 x = np.arange(xstart, xstop, increment)
9
10 y = x**2;
11
12
13 # Exact/Analytical Solution
14 dydx_exact = 2*x
15
16 plt.plot(x, dydx_exact, 'o-')
17
18
19
20
21 # Exact/Analytical Solution

```

```

22 dydx_num = np.diff(y) / np.diff(x);
23
24
25 xstart = -2
26 xstop = 2
27
28 x = np.arange(xstart, xstop, increment)
29
30
31 plt.plot(x, dydx_num, 'o-')

```

Listing 30.3: Numerical Differentiation

This gives the the plot shown in Figure 30.4.

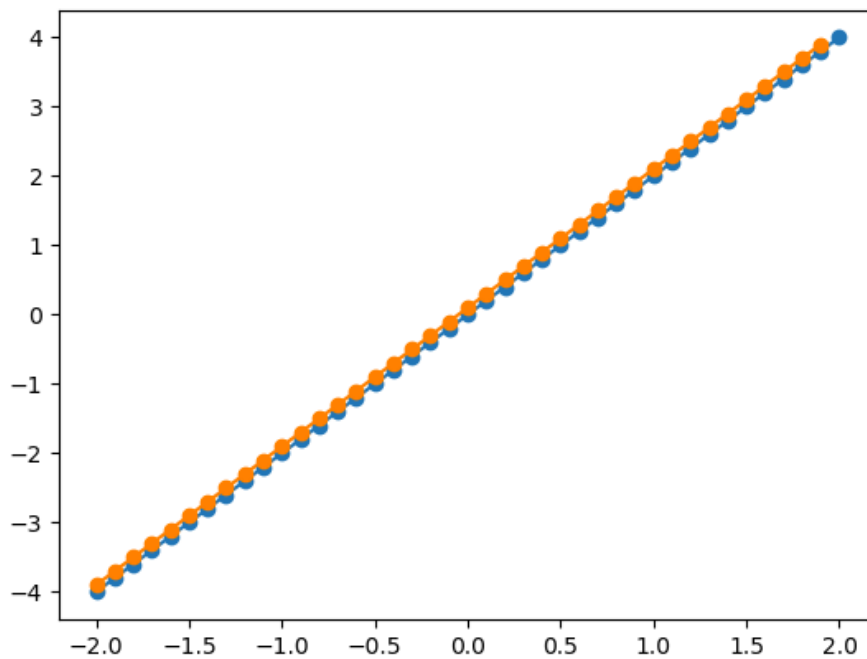


Figure 30.4: Numerical Differentiation Example

We see that there numeric solutions becomes very close to the exact solutions. When $h \rightarrow 0$ we should expect that the numerical solutions should exactly match the exact solutions.

[End of Example]

Exercise 30.0.1. Numerical Differentiation

Given the following equation:

$$y = x^3 + 2x^2 - x + 3 \quad (30.10)$$

Find $\frac{dy}{dx}$ analytically (use "pen and paper").

Define a vector x from -5 to +5 and use the **diff** function to approximate the derivative y with respect to x ($\Delta y / \Delta x$).

Compare the data in a 2D array and/or plot both the exact value of $\frac{dy}{dx}$ and the approximation in the same plot.

Increase number of data point to see if there are any difference.

Do the same for the following functions:

$$y = \sin(x) \quad (30.11)$$

$$y = x^5 - 1 \quad (30.12)$$

[End of Exercise]

30.1 Differentiation on Polynomials

A polynomial is expressed as:

$$p(x) = p_1 x^n + p_2 x^{n-1} + \dots + p_n x + p_{n+1} \quad (30.13)$$

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

The differentiation of the polynomial $p(x)$ will be:

$$\frac{dp(x)}{dx} = p_1 n x^{n-1} + p_2 (n-1) x^{n-2} + \dots + p_n \quad (30.14)$$

Performing differentiation on polynomials is easy because you only need to use the basic differentiation rules:

$$f(x) = ax^n \rightarrow f'(x) = a \cdot n \cdot x^{n-1} \quad (30.15)$$

$$f(x) = C \rightarrow f'(x) = 0 \quad (30.16)$$

Where C is a constant.

Examples:

$$f(x) = 4x^3 \rightarrow f'(x) = 12x^2 \quad (30.17)$$

$$f(x) = 5 \rightarrow f'(x) = 0 \quad (30.18)$$

We could easily made a general function in Python that could handle this. If you dont want to do that, you can use the `polyder()` NumPy function.

Example 30.1.1. Differentiation on Polynomials

Given the polynomial

$$p(x) = 2 + x^3 \quad (30.19)$$

We can rewrite the polynomial like this:

$$p(x) = 1 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x + 2 \quad (30.20)$$

We know that: $p'(x) = 3x^2$

The polynomial is defined in Python as:

```
1 import numpy as np
2
3 p = [1, 0, 0, 2]
4
5 dpdx = np.polyder(p)
6
7 print(dpdx)
```

Listing 30.4: Differentiation on Polynomials

The results:

`dpdx = [3 0 0]`

Which is correct, because

$$p'(x) = 3 \cdot x^2 + 0 \cdot 2x + 0 \cdot 1 \quad (30.21)$$

with the coefficients: $p_1 = 3, p_2 = 0, p_3 = 0$
And this is written as an array `[3 0 0]` in Python.

The Python code can also be written like this:

```

1 import numpy as np
2
3 p = np.poly1d([1,0,0,2])
4 print(p)
5
6 dpdx = np.polyder(p)
7 print(dpdx)

```

Listing 30.5: Differentiation on Polynomials

Here we have used the **poly1d()** function in the NumPy library. Make sure to try this example!

[End of Example]

Exercise 30.1.1. Differentiation on Polynomials

Consider the following equation:

$$y = x^3 + 2x^2 - x + 3 \quad (30.22)$$

Use differentiation on the Polynomial to find $\frac{dy}{dx}$

[End of Exercise]

Exercise 30.1.2. Differentiation on Polynomials

Find the derivative for the product:

$$y = (3x^2 + 6x + 9)(x^2 + 3x) \quad (30.23)$$

i.e, you need to find $\frac{dy}{dx}$

[End of Exercise]

x	y
-2	4
-1	1
0	0
1	1
2	4

Table 30.1: Example Data

Chapter 31

Numerical Integration

The integral of a function $f(x)$ is denoted as:

$$\int_a^b f(x)dx \quad (31.1)$$

An integral can be seen as the area under a curve.

Given $y = f(x)$ the approximation of the Area (A) under the curve can be found dividing the area up into rectangles and then summing the contribution from all the rectangles:

$$\sum_{i=1}^{n-1} (x_{i+1} - x_i) \cdot (y_{i+1} + y_i)/2 \quad (31.2)$$

This is known as the Trapezoid rule.

We approximate the integral by using n trapezoids formed by using straight line segments between the points (x_{i-1}, y_{i-1}) and (x_i, y_i) for $1 \leq i \leq n$ as shown in Figure 31.1.

The area of a trapezoid is obtained by adding the area of a rectangle and a triangle (31.3):

$$A = y_0 \Delta x + \frac{1}{2}(y_1 - y_0) \Delta x = \frac{(y_0 + y_1) \Delta x}{2} \quad (31.3)$$

See Figure 31.2.

Example 31.0.1. Basic Integral Example

Given the function:

$$y = x^2 \quad (31.4)$$

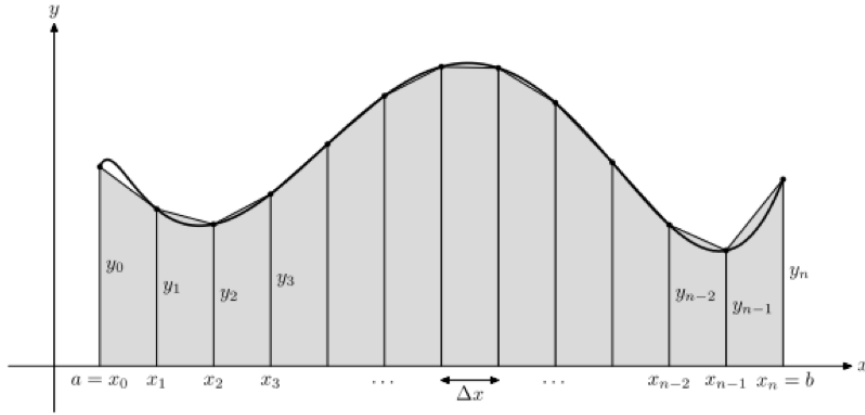


Figure 31.1: Numerical Integral Approach

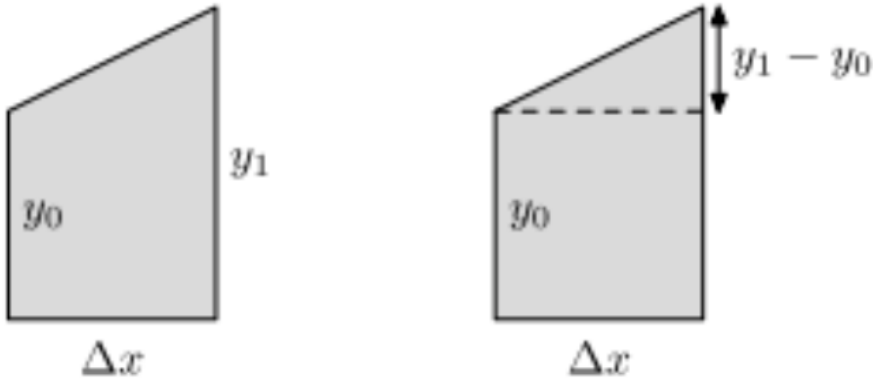


Figure 31.2: The Area of a Trapezoid

We know that the exact solution is:

$$\int_a^b x^2 dx = \frac{a^3}{3} \quad (31.5)$$

The integral from 0 to 1 is:

$$\int_0^1 x^2 dx = \frac{1}{3} \approx 0.3333 \quad (31.6)$$

We will use the trapezoid rule and the diff function in MATLAB to solve the numerical integral of x^2 from 0 to 1.

The Python code implements the formula known as the Trapezoid rule:

$$I = \int_a^b f(x) dx \approx \frac{1}{2} \sum_{i=1}^n (f(x_i) + f(x_{i+1})) \cdot (x_i - x_{i-1}) \quad (31.7)$$

Assuming that we use N sub intervals in order to calculate the approximation we get:

$$A = \frac{\Delta x}{2} \sum_{i=1}^N (f(x_i) + f(x_{i+1})) \quad (31.8)$$

Where

$$\Delta x = \frac{b-a}{N} \quad (31.9)$$

The Python code for this is:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 a = 0
5 b = 1
6 N = 10
7
8
9 x = np.linspace(a,b,N+1)
10
11 y = x**2;
12
13 y_right = y[1:]
14 y_left = y[:-1]
15
16
17 # Trapezoid Rule
18 dx = (b - a)/N
19 A = (dx/2) * np.sum(y_right + y_left)
20
21 print(A)
22
23
24 plt.plot(x,y)
25 plt.xlim([0,1]); plt.ylim([0,1]);

```

Listing 31.1: Trapezoid Rule

We get the following results:

$A = 0.3350000000000001$

Which is a good approximation when we now the exact answer is $A = \frac{1}{3}$

[End of Example]

In the example above we implemented our own integration from scratch. That's always a good approach because then we get to know the mathematics behind. But typically you want to use a predefined function that do the job for you.

The SciPy subpackage `scipy.integrate` contains several functions for approximating definite integrals numerically.

Take a closer look at the documentation in order to get an overview of the different functions that are available.

<https://docs.scipy.org/doc/scipy/reference/integrate.html#module-scipy.integrate>

Example 31.0.2. Integration using the quad function

In this example we will use the **quad** function in the SciPy subpackage `scipy.integrate`.

See the following for detailed information about the **quad** function:

<https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html#scipy.integrate.quad>

Python code:

```
1 from scipy import integrate
2
3 a = 0
4 b = 1
5
6 def y(x):
7     return x**2
8
9 I = integrate.quad(y, a, b)
10
11 print(I)
```

Listing 31.2: xxx

We get the following results:

`I = (0.3333333333333337, 3.700743415417189e-15)`

Which is a good approximation when we know the exact answer is $A = \frac{1}{3}$

The second result is an estimate of the absolute error in the given result.

[End of Example]

Example 31.0.3. Integration using the builtin trapz function

In this example we will use the builtin **trapz** function in SciPy.

See the following for detailed information about the **trapz** function:

<https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.trapz.html#scipy.integrate.trapz>

Python code:

```
1 import numpy as np
2
3 a = 0
4 b = 1
5 N = 10
6 dx = (b - a)/N
```

```

7
8 x = np.linspace(a,b,N+1)
9
10 y = x**2;
11
12 I = np.trapz(y,x,dx)
13
14 print(I)

```

Listing 31.3: xxx

We get the following results:

$I = 0.33499999999999996$

Which is a good approximation when we know the exact answer is $A = \frac{1}{3}$

[End of Example]

Exercise 31.0.1. Numerical Integration

Use some of the integral approximations shown in the examples above on the following equation:

$$y = x^3 + 2x^2 - x + 3 \quad (31.10)$$

Find the integral of y with respect to x , evaluated from -1 to 1

Compare the different methods.

You should also find the exact solution using "pen and paper for comparison. Compare the result with the exact solution.

[End of Exercise]

Exercise 31.0.2. Numerical Integration 2

Use some of the integral approximations shown in the examples above on the following equation:

$$y = \sin(x) \quad (31.11)$$

Find the integral of y with respect to x , evaluated from -1 to 1

Compare the different methods.

You should also find the exact solution using "pen and paper for comparison. Compare the result with the exact solution.

[End of Exercise]

Exercise 31.0.3. Numerical Integration 3

Use some of the integral approximations shown in the examples above on the following equation:

$$y = x^5 - 1 \quad (31.12)$$

Find the integral of y with respect to x, evaluated from -1 to 1

Compare the different methods.

You should also find the exact solution using "pen and paper for comparison. Compare the result with the exact solution.

[End of Exercise]

31.1 Integration on Polynomials

A polynomial is expressed as:

$$p(x) = p_1x^n + p_2x^{n-1} + \dots + p_nx + p_{n+1} \quad (31.13)$$

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

In Python we can use the **polyint** function to perform integration on polynomials. This function works the same way as the **polyder** function which performs differentiation on polynomials.

Exercise 31.1.1. Integration on Polynomials

Given the following equation:

$$y = x^3 + 2x^2 - x + 3 \quad (31.14)$$

Find the integral for this function using polyint.

You should also find the exact solution using "pen and paper for comparison.

You should also plot the function in the interval [a,b] where you choose different values for a and b.

You should also find the integral for these intervals.

[End of Exercise]

31.2 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 31.2.1. Piston Cylinder

Given the following piston cylinder shown in Figure 31.3.

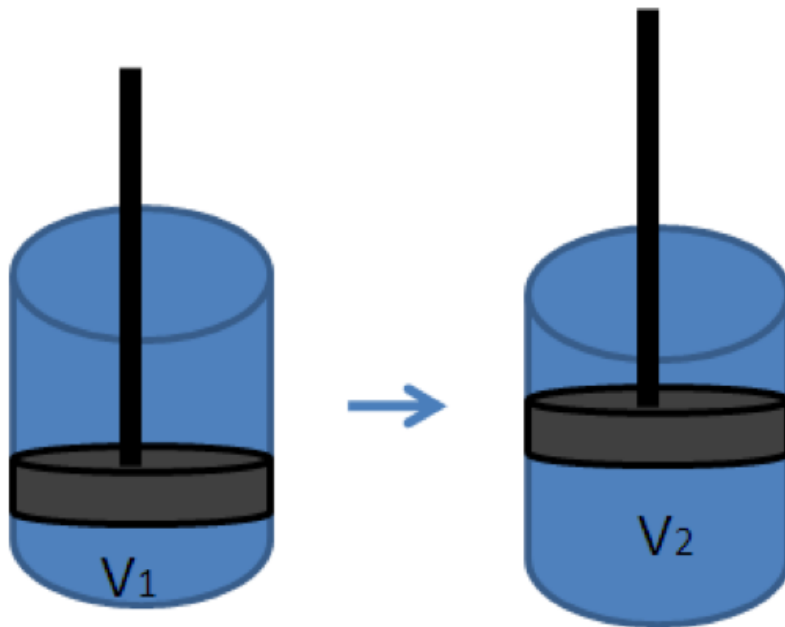


Figure 31.3: Piston Cylinder

Find the work (W) produced in a piston cylinder device by solving the following equation:

$$W = \int_{V_1}^{V_2} P dV \quad (31.15)$$

Assume that the ideal gas law applies:

$$PV = nRT \quad (31.16)$$

Where:

- P = Pressure

- V = Volume [m^3]
- n = number of moles [$kmol$]
- R = universal gas constant [$8.314kJ/kmolK$]
- T = Temperature, [K]

We also assume that the piston contains 1 mol of gas at 300K and that the temperature is constant during the process. $V_1 = 1m^3, V_2 = 5m^3$

Implement the integral using some of the different methods shown in previous examples. You should also try to find the exact solution using "pen and paper" for comparison.

[End of Exercise]

Chapter 32

Optimization

Optimization is important in mathematics, control and simulation applications. Optimization is based on finding the minimum of a given criteria function.

Example 32.0.1. Basic Optimization

Given the following function:

$$f(x) = x^2 + 2x + 1 \quad (32.1)$$

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 32.1: Optimization in Python

This gives the the plot shown in Figure 32.1.

We will use **fminbound** to find the minimum of the function.

Python code:

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

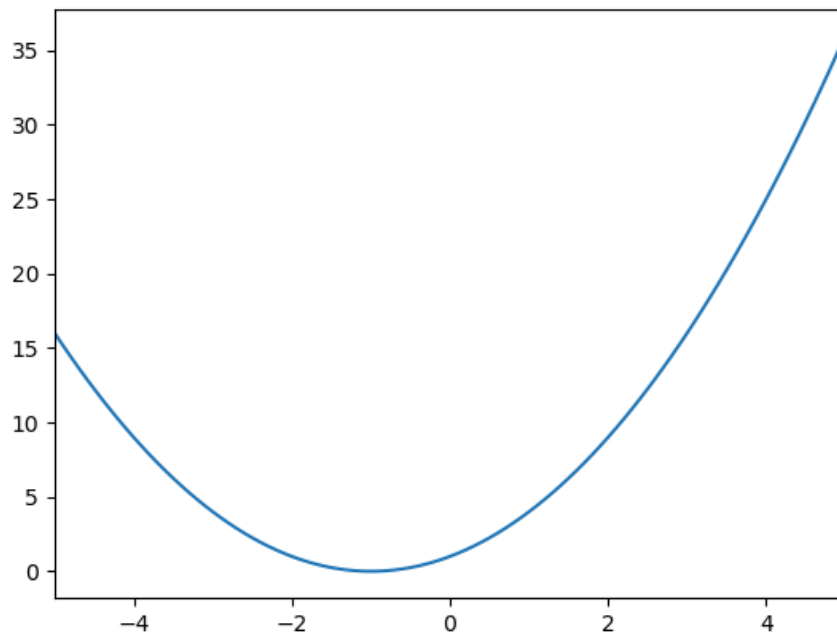



Figure 32.1: Optimization Example

```

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 32.2: xxx

The result becomes:

xmin = -1.0

We see that this is the correct answer based on our plot in Figure 32.1.

[End of Example]

Exercise 32.0.1. Make your own Minimization function

Given the following function:

$$f(x) = x^2 + 2x + 1 \quad (32.2)$$

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 32.0.2. Optimization Functions in Python

Given the following function:

$$f(x) = x^2 + 2x + 1 \quad (32.3)$$

Test the different optimization (finding minimum) functions:

`scipy.optimize.fmin`

`scipy.optimize.minimize_scalar`

`scipy.optimize.minimize`

Compare the results.

[End of Exercise]

Exercise 32.0.3. Optimization

Given the following function:

$$f(x) = x^3 - 4x \quad (32.4)$$

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 32.0.4. Minimum for function with 2 variables

Given the following function:

$$f(x, y) = 2(x - 1)^2 + x - 2 + (y - 2)^2 + y \quad (32.5)$$

Plot the function and find the minimum.

[End of Exercise]

Exercise 32.0.5. Optimization - Rosenbrock's Banana Function

Given the following function:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \quad (32.6)$$

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 33

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 34

Differential Equations

34.1 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:

$$\frac{dy}{dt} = f(t, y), y(t_0) = y_0 \quad (34.1)$$

Note! Different notation is used: $\frac{dy}{dt} = y' = \dot{y}$

This document will use these different notations interchangeably.

Example 34.1.1. Example of Dynamic Systems

Given the following differential equation:

$$\dot{x} = -ax + bu \quad (34.2)$$

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 have we can solve these differential equations in many different ways.

We can use so-called ODE solvers or we can make discrete version of the differential equations using discretization methods like Euler, etc.

With ODE solvers Python can solve these equations numerically. Higher order differential equations must be reformulated into a system of first order differential equations.

In chapter 35 we will simulate (solve and plot the results) such differential equations numerically using Euler discretization.

Chapter 35

Discrete Systems

When dealing with computers we need to deal with discrete systems.

35.1 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 $T_s = 0.1s$.

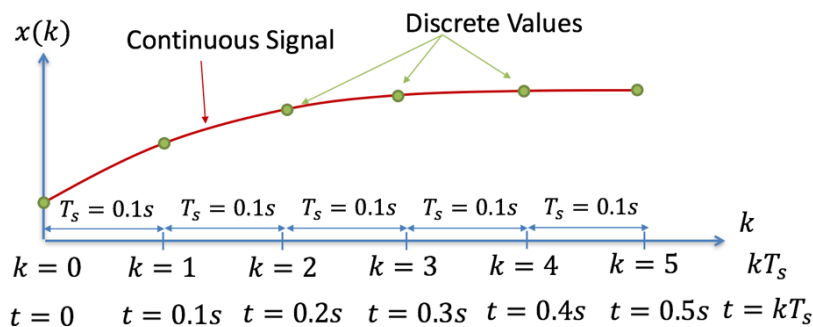


Figure 35.1: Discretization

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

A simple discretization method is the Euler Forward method:

$$\dot{x} = \frac{x(k+1) - x(k)}{T_s} \quad (35.1)$$

T_s is the Sampling Time

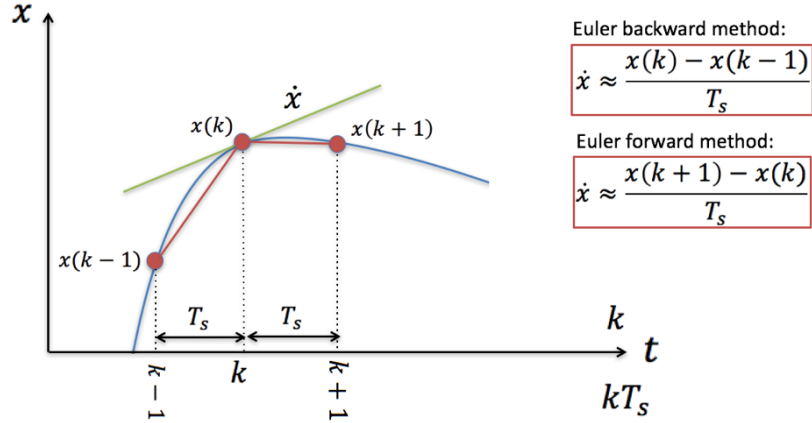


Figure 35.2: Euler Discretization methods

Lots of other discretization methods do exists, such as “Euler backward”, Zero Order Hold (ZOH), Tustin’s method, etc.

Example 35.1.1. Simulation of Discrete System

Given the following differential equation:

$$\dot{x} = -ax + bu \quad (35.2)$$

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:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s} \quad (35.3)$$

T_s - Sampling Interval

Then we get:

$$\frac{x_{k+1} - x_k}{T_s} = -ax_k + bu_k \quad (35.4)$$

This gives the following discrete differential equation:

$$x_{k+1} = (1 - T_s a)x_k + T_s b u_k \quad (35.5)$$

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 35.1: Simulation of Discrete Dynamic System in Python

The simulation gives the results as shown in Figure 35.3.

Lets also take a look at the Variable Explorer as shown in Figure 35.4.

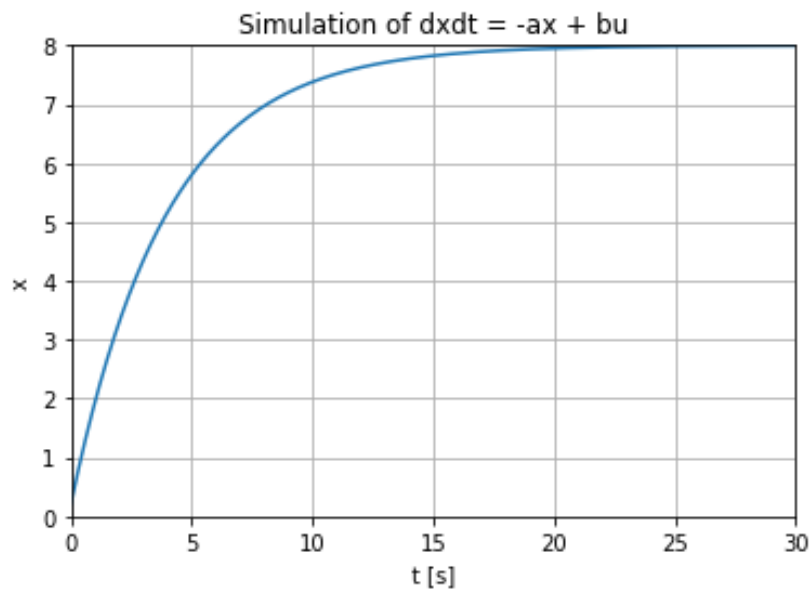


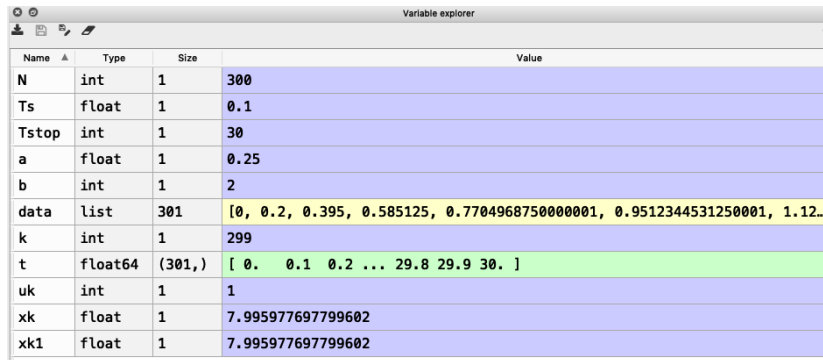
Figure 35.3: Simulation of Discrete System using Python

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
18 # Simulation
19 for k in range(N+1):
20     x[k+1] = (1 - a*Ts) * x[k] + Ts * b * uk
21
22
23 # Plot the Simulation Results
24 t = np.arange(0, Tstop+2*Ts, Ts) #Create the Time Series
25
26 plt.plot(t, x)
27
28 # Formatting the appearance of the Plot
29 plt.title('Simulation of dxdt = -ax + bu')
30 plt.xlabel('t [s]')
31 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 35.4: Variable Explorer for Discrete Simulation Example

```

32 plt.grid()
33 plt.axis([0, 30, 0, 8])
34 plt.show()

```

Listing 35.2: 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]

35.2 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 35.2.1. 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:

$$\text{birth rate} = bx \quad (35.6)$$

$$\text{death rate} = px^2 \quad (35.7)$$

Then the total rate of change of bacteria population is:

$$\dot{x} = bx - px^2 \quad (35.8)$$

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.

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 35.2.2. Simulation with 2 variables

Given the following system:

$$\frac{dx_1}{dt} = -x_2 \quad (35.9)$$

$$\frac{dx_2}{dt} = x_1 \quad (35.10)$$

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 36

Real-Time Simulations

36.1 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 35.

Example 36.1.1. Basic Simulation of Discrete System

We will simulate the discrete system given as follows:

$$x_{k+1} = (1 - T_s a)x_k + T_s b u_k \quad (36.1)$$

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 36.1: Simulation of Discrete Dynamic System in Python

The simulation gives the results as shown in Figure 36.1.

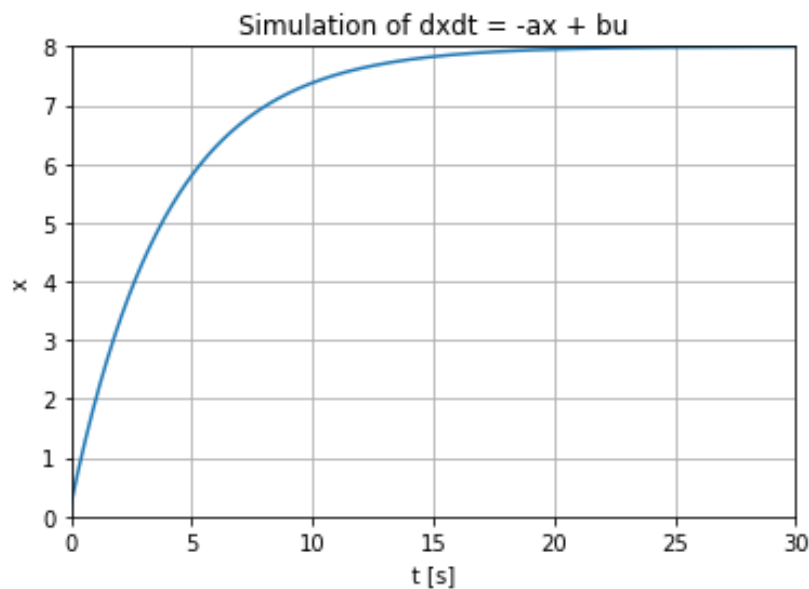


Figure 36.1: Simulation of Discrete System using Python

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.

36.2 Introduction to Real-Time Plotting

You can also use the matplotlib for real-time plotting.

Example 36.2.1. 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 36.2: Real-Time Plotting in Python

We get the following plot as shown in Figure 37.1.

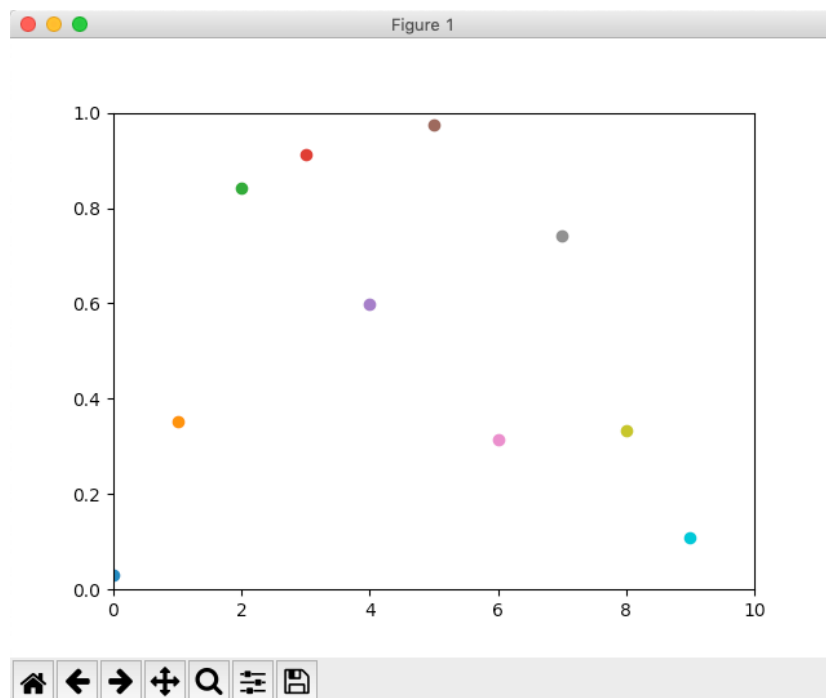


Figure 36.2: Real-Time Plotting with Python

You cannot see the the actual behavior of the plot by watching Figure 37.1, so

you need to run the Python program yourself.

If you run the code you see the plot is updated with a new value every second as specified in the code.

[End of Example]

Note! If you use Anaconda and Spider, you typically need to change the the settings for how graphics are are displayed in Spyder.

Select Preferences from the menu, then IPython console in the list of categories on the left, then the tab Graphics at the top, and change the Graphics back-end from Inline to e.g. Automatic or Qt. See Figure 37.2.

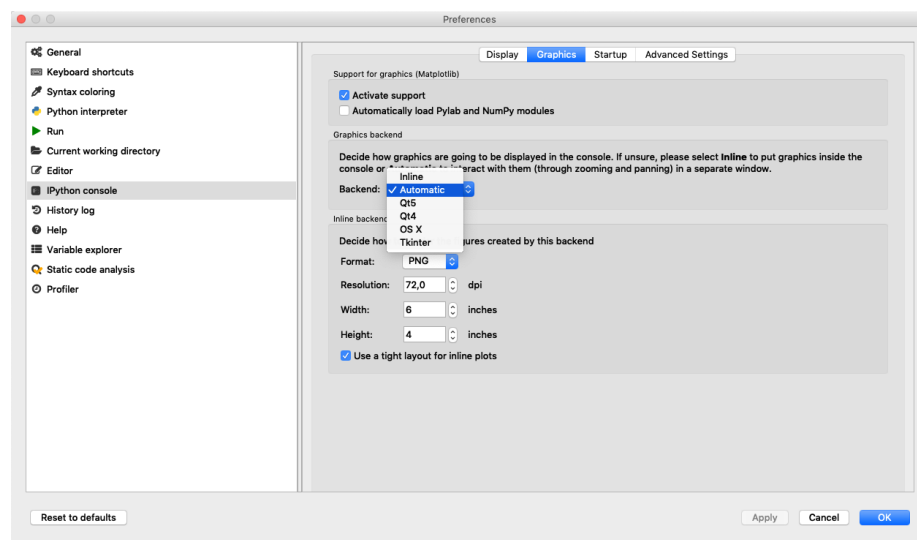


Figure 36.3: Change how Graphics are displayed in the Spyder Editor

Example 36.2.2. 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 36.3: Real-Time Simulation of Discrete Dynamic System in Python

Figure 36.4 shows the Real-Time Plot for this example.

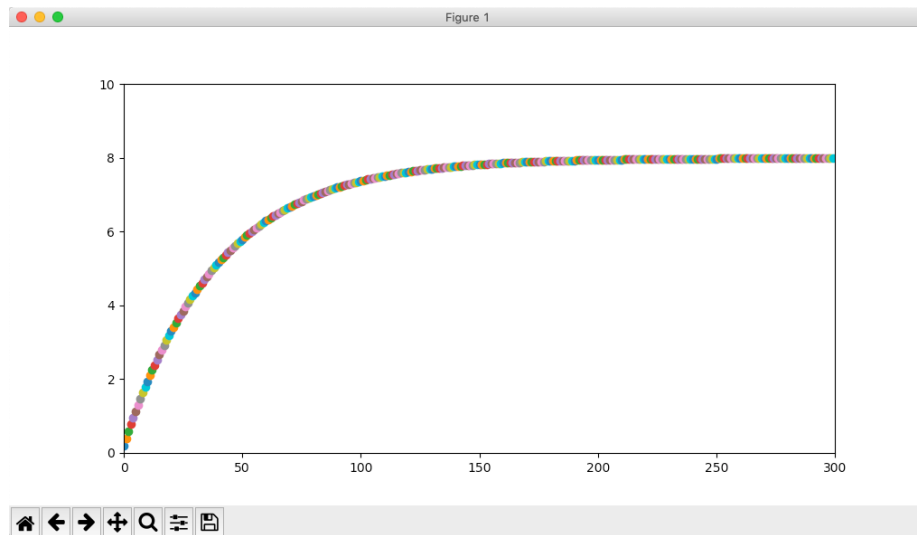


Figure 36.4: Real-Time Simulation of Discrete System

You cannot see the the actual behavior of the plot by watching Figure 36.4, so you need to run the Python program yourself.

If you run the code you see the plot is updated with a new value every second as specified in the code.

In this example we have locked the scaling using the axis method. If we remove or comment out the line "plt.axis([0, N, 0, 10])", we get a plot the automatically scales the x axis and the y axis. Whats best depends on if you know the simulation length from the beginning and know the minimum and maximum value of the simulation results.

Figure 36.5 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 36.5, so you need to run the Python program yourself.

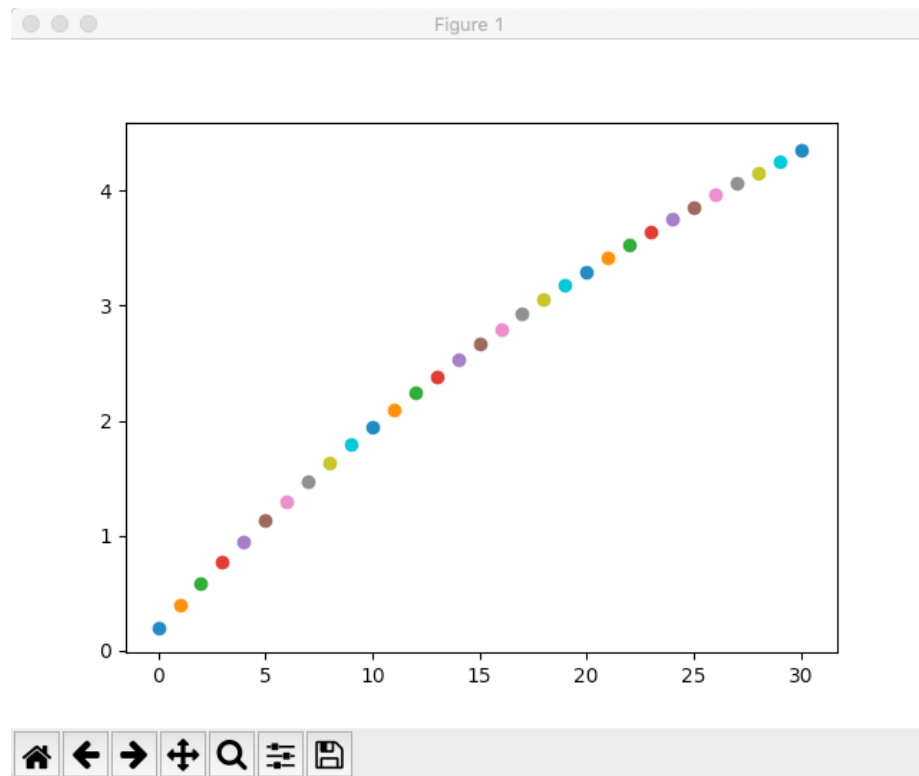


Figure 36.5: Real-Time Simulation of Discrete System with Auto-scaling

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
21 for k in range(N):
22     xk1 = (1 - a*Ts) * xk + Ts * b * uk
23     xk = xk1
24     data.append(xk1)
25
26     t = Ts*k
27
28     plt.scatter(t, xk1)
29     plt.pause(Ts)
30
31 plt.show()

```

Listing 36.4: Real-Time Simulation of Discrete Dynamic System in Python

Figure 36.6 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 36.6, so you need to run the Python program yourself.

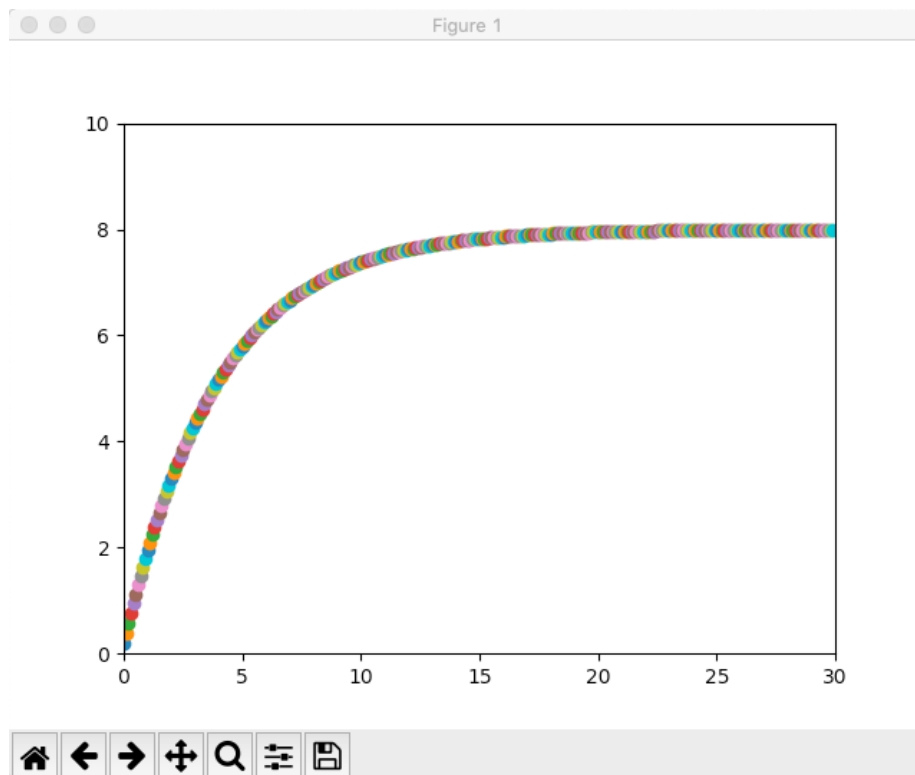


Figure 36.6: Real-Time Simulation of Discrete System

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 36.5: Real-Time Simulation of Discrete Dynamic System in Python

Run the example and notice the difference.

[End of Example]

36.3 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 36.3.1. 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 FuncAnimation 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 36.6: Real-Time Plotting with Animation

Figure 37.3 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 37.3, so you need to run the Python program yourself.

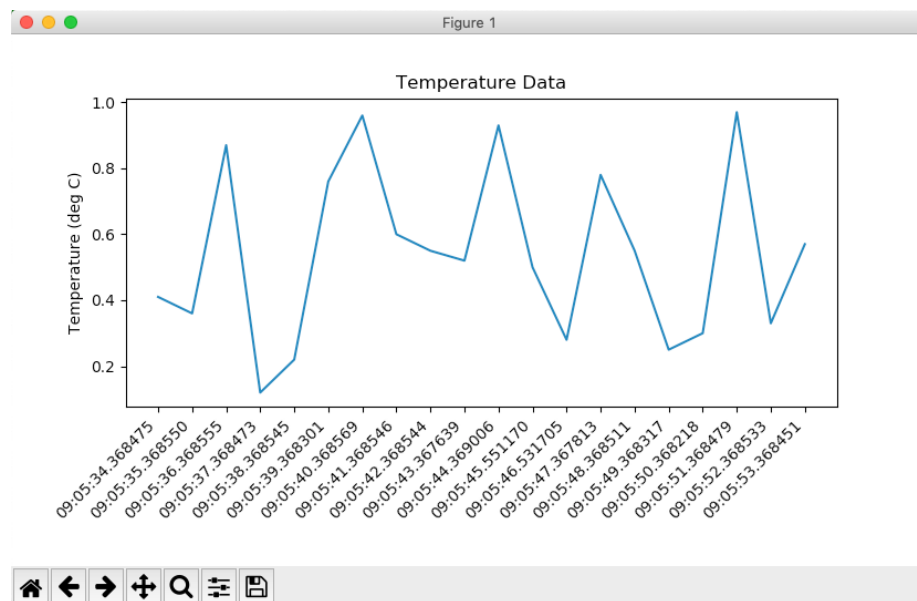


Figure 36.7: Real-Time Plotting with Animation

[End of Example]

Example 36.3.2. 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 global 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 36.7: Real-Time Simulation of Discrete Dynamic System with Animation

Figure 36.8 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 36.8, so you need to run the Python program yourself.

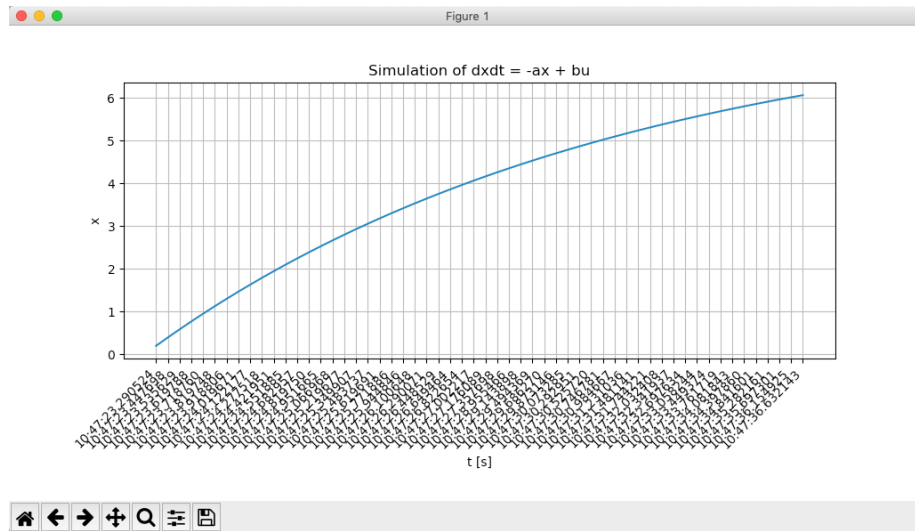


Figure 36.8: Real-Time Simulation of Discrete Dynamic System with Animation

[End of Example]

36.3.1 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 36.3.3. 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 36.8: Real-Time Plotting with Animation

Figure 37.4 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 37.4, so you need to run the Python program yourself.

[End of Example]

Example 36.3.4. Discrete Simulations with Animation with improved Performance

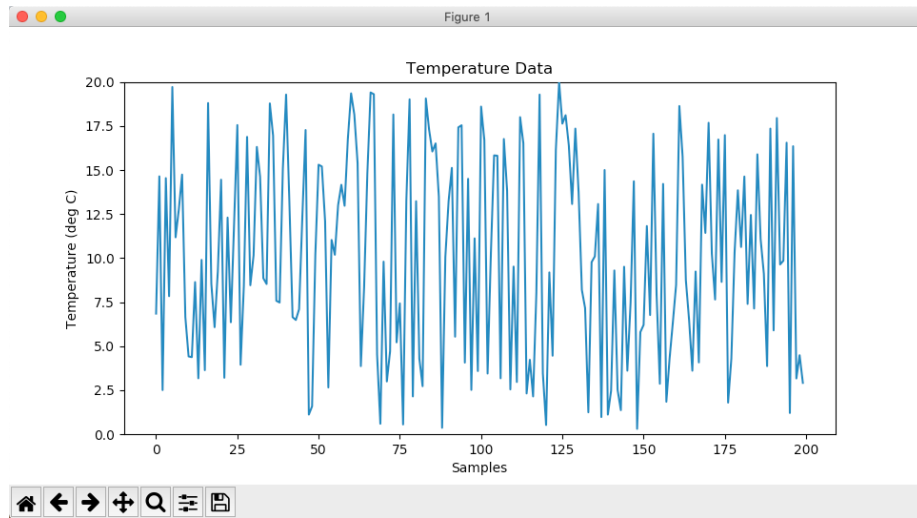


Figure 36.9: Real-Time Plotting with Animation

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
           a global variable
38
39     # Model of Discrete System
40     xk1 = (1 - a*Ts) * xk + Ts * b * uk
41     xk = xk1
42
43     # Add y to list
44     ys.append(xk1)
45
46     # Limit y list to set number of items
47     ys = ys[-x_len:]
48
49     # Update line with new Y values
50     line.set_ydata(ys)
51
52     return line ,
53
54 # Set up plot to call animate() function periodically
55 ani = animation.FuncAnimation(fig ,
56     simulation ,
57     fargs=(ys,) ,
58     interval=100,
59     blit=True)
60 plt.show()

```

Listing 36.9: Real-Time Simulation of Discrete Dynamic System with Animation and increased Speed

Figure 36.10 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 36.10, 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

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>

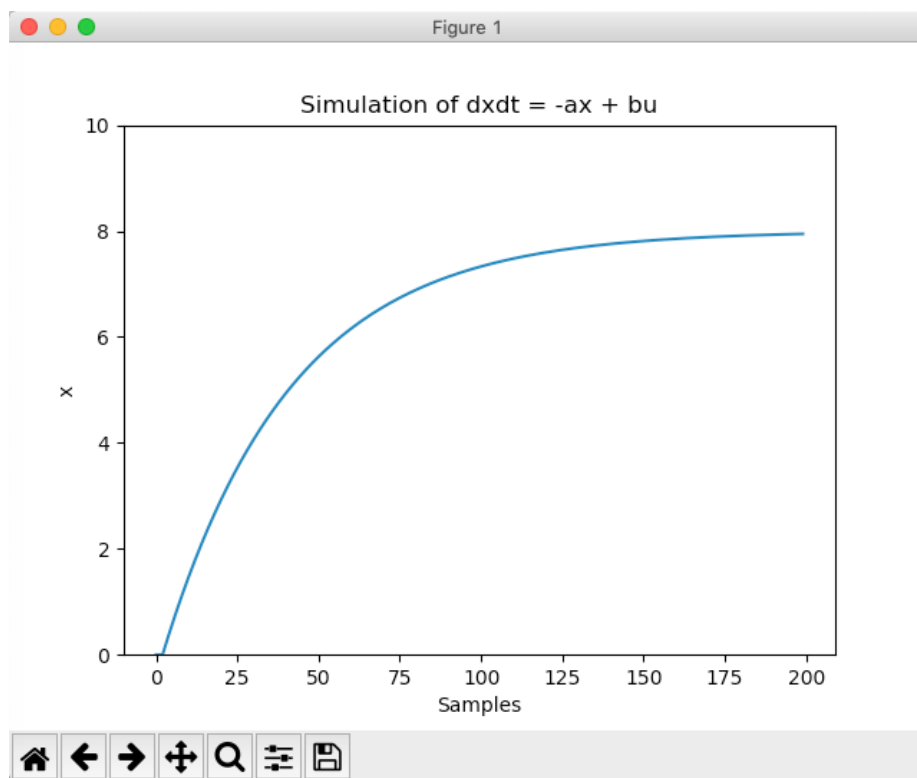


Figure 36.10: Real-Time Plotting with Animation

Part VII

Data Acquisition (DAQ)
with Python

Chapter 37

Plotting Sensor Data

37.1 Introduction

Typically we want to plot the data from the sensor. We can plot save the data in an array and then plot the data at the end of the program, but more likely we want to plot one value at the time inside the loop, so-called "Real-Time plotting".

In this chapter we only show how you can plot the data from any given sensor using this general approach. Instead of the actual sensor data we just use the random generator in Python.

To read the actual sensor data you typically need a DAQ (Data Acquisition) device connected to you PC or, e.g, a Raspberry Pi device. In all cases you will typically need to install a driver from the vendor of the DAQ device or the sensor you are using.

37.2 Introduction to Real-Time Plotting

You can also use the matplotlib for real-time plotting.

Example 37.2.1. 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 37.1: Real-Time Plotting in Python

We get the following plot as shown in Figure 37.1.

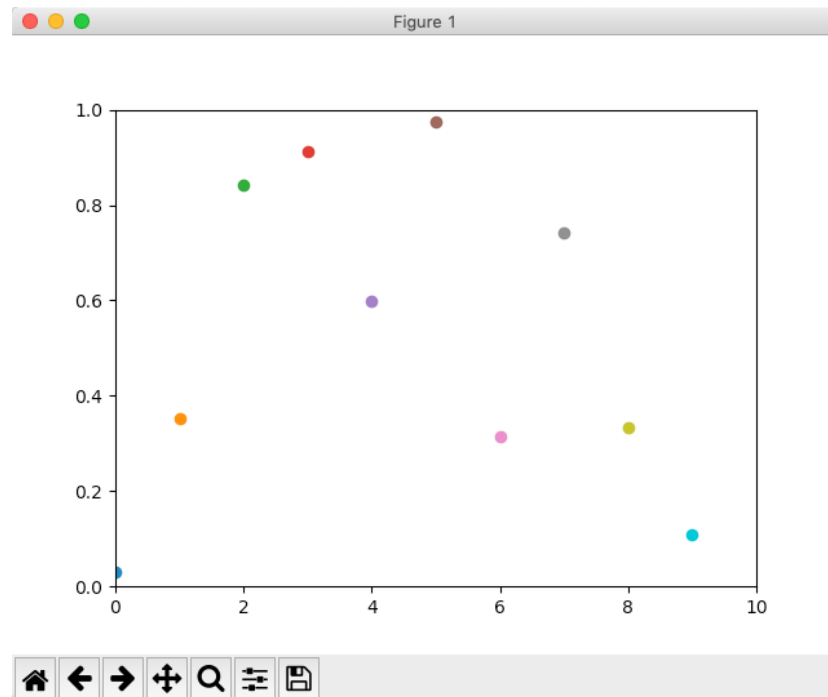


Figure 37.1: Real-Time Plotting with Python

You cannot see the the actual behavior of the plot by watching Figure 37.1, 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 37.2.

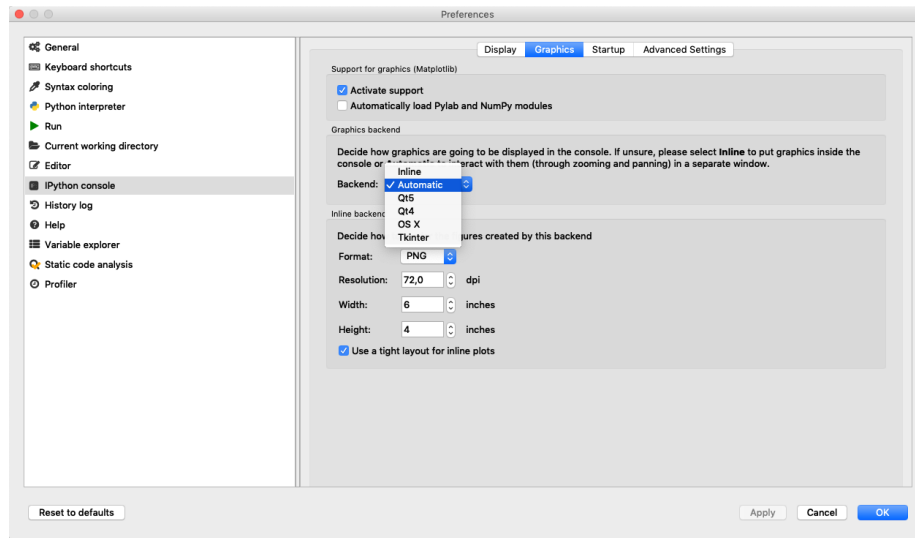


Figure 37.2: Change how Graphics are displayed in the Spyder Editor

37.3 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 37.3.1. 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 FuncAnimation 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 37.2: Real-Time Plotting with Animation

Figure 37.3 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 37.3, so you need to run the Python program yourself.

[End of Example]

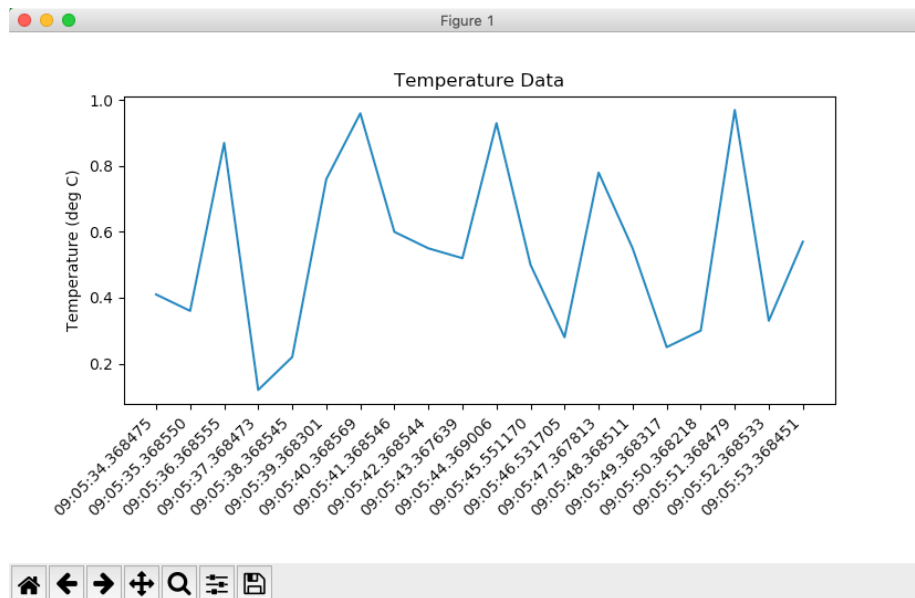


Figure 37.3: Real-Time Plotting with Animation

37.3.1 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 37.3.2. 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 37.3: Real-Time Plotting with Animation

Figure 37.4 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 37.4, 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

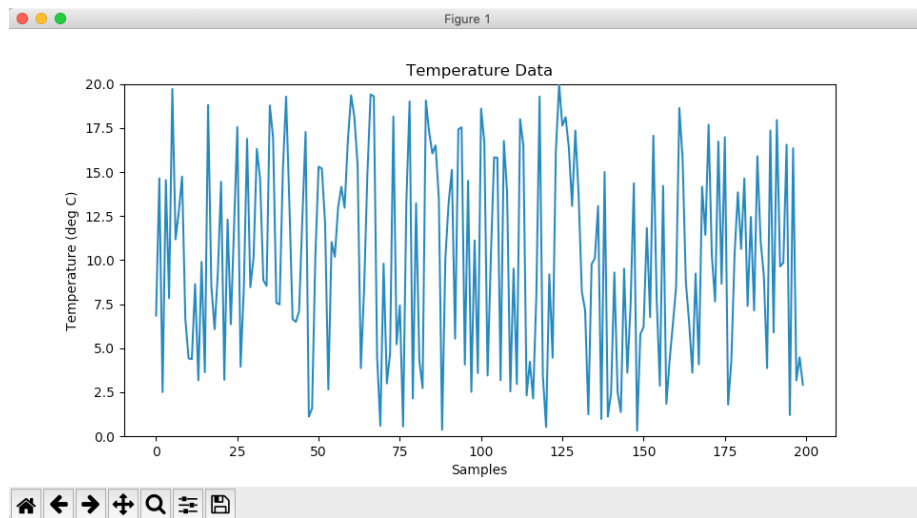


Figure 37.4: Real-Time Plotting with Animation

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 38

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.

38.1 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 38.1 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/>

38.2 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.

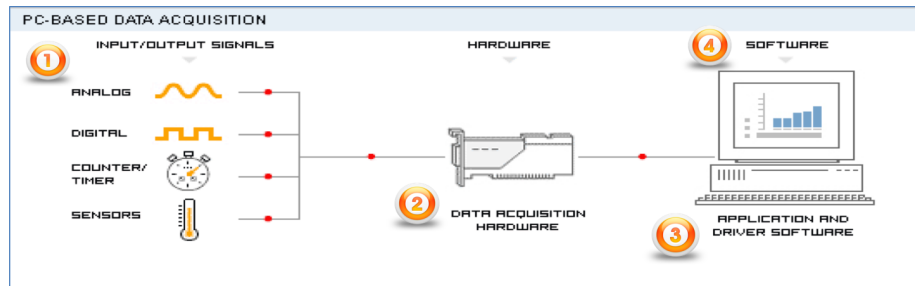


Figure 38.1: Data Acquisition (DAQ) System

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 38.2 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 38.2: USB-6008

For more information about NI DAQ:
ni.com/daq

For more information about Python Resources for NI Hardware and Software:
ni.com/python

Another option is to use the **PyDAQmx** Python package. This package allows users to use data acquisition hardware from National Instrument with python. It makes an interface between the NIDAQmx driver and python. It currently works only on Windows. The package is not an open source driver from NI acquisition hardware. You first need to install the driver provided by NI.

Web:
<https://pypi.org/project/PyDAQmx/>

<https://pythonhosted.org/PyDAQmx/>

38.2.1 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>

38.2.2 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 38.3 shows Measurement Automation Explorer (MAX).

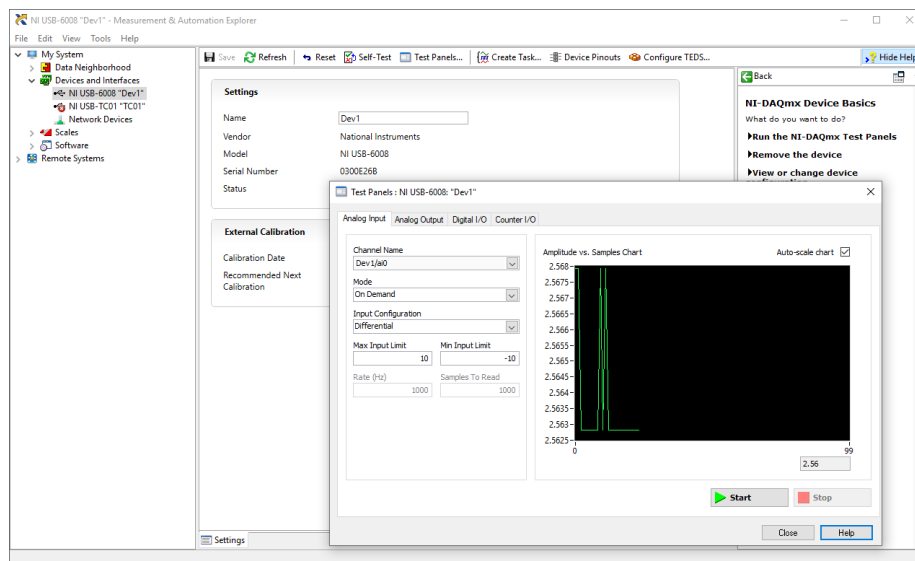


Figure 38.3: Measurement Automation Explorer(MAX)

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.

38.3 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 these 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.

38.3.1 Analog Write

Example 38.3.1. 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 38.1: Analog Write using NI DAQ Device

Note! The USB-6008 can only output a voltage signal between 0 and 5V.

[End of Example]

38.3.2 Analog Read

Example 38.3.2. 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 38.2: Analog Read using NI DAQ Device

[End of Example]

Example 38.3.3. 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 38.3: Analog Read with RSE

[End of Example]

Example 38.3.4. 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 38.4: Analog Read with Differential

[End of Example]

38.3.3 Digital Write

Example 38.3.5. 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 38.5: Digital Write using NI DAQ Device

[End of Example]

38.3.4 Digital Read

Example 38.3.6. 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 38.6: Digital Read using NI DAQ Device

[End of Example]

You should use the "nidaqmx.stream_readers" and nidaqmx.stream_writers classes to increase the performance of your application, which accept pre-allocated NumPy arrays.

https://nidaqmx-python.readthedocs.io/en/latest/stream_readers.html#module-nidaqmx.stream_readers

https://nidaqmx-python.readthedocs.io/en/latest/stream_writers.html#module-nidaqmx.stream_writers

38.4 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 38.4 shows an overview of LEDs.

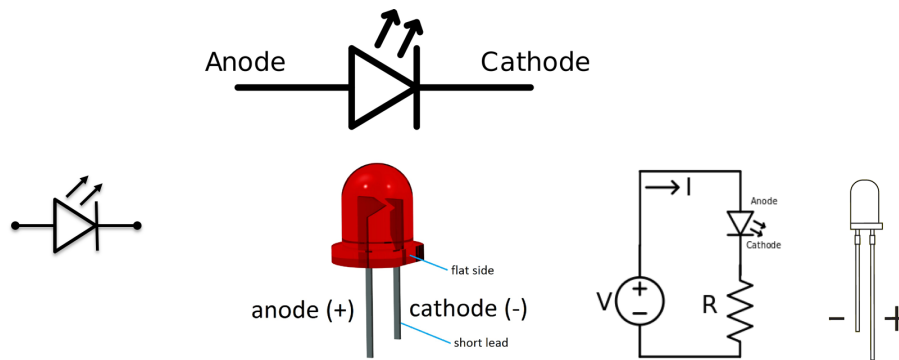


Figure 38.4: LED Overview

A breadboard is used to wire electric components together. Figure 38.5 shows how you should wire a LED using a Breadboard.

Figure 38.6 shows how you wire the LED and connect it to the DAQ device.

Python code for turning on the LED

Example 38.4.1. 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 38.7: Turn on a LED using Python

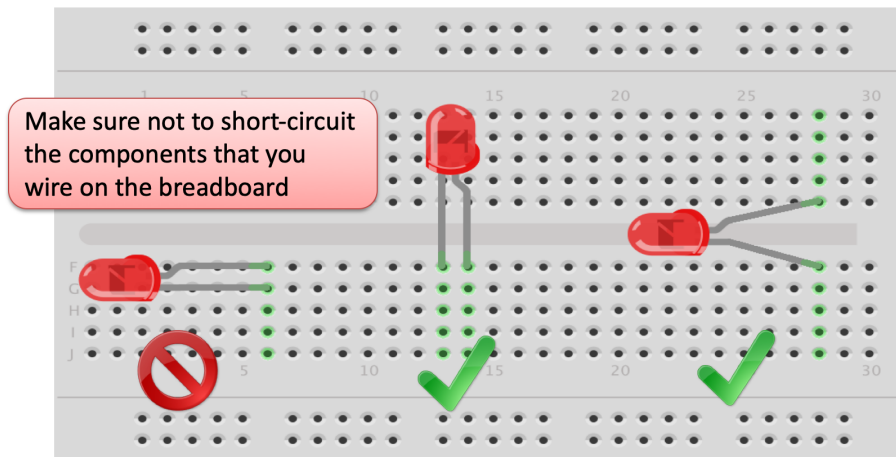


Figure 38.5: How to wire a LED on a Breadboard

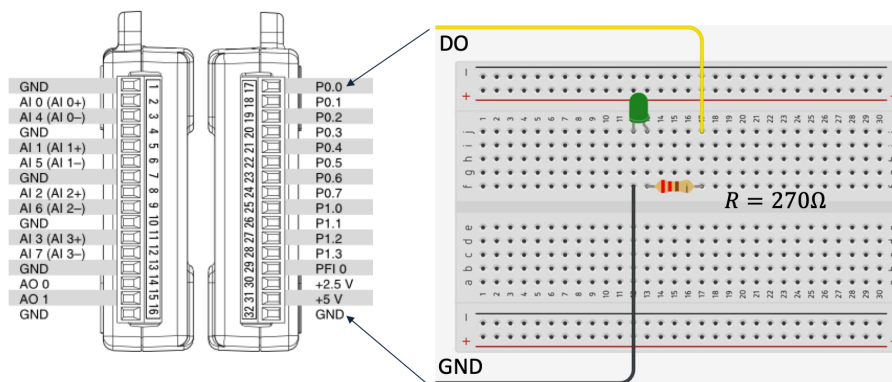


Figure 38.6: Wire the LED and connect to the DAQ device

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 38.8: Controlling a LED using Python

[End of Example]

38.5 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.

38.5.1 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 38.7 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 38.8 show how we can wire the TMP36 together with the USB-6008 DAQ device.

Figure 38.9 shows the TMP3x Datasheet.

We need to convert form Voltage (V) to degrees Celsius.

From the Datasheet (Figure 38.9) we have:

$$(x_1, y_1) = (0.75V, 25^\circ) \quad (38.1)$$

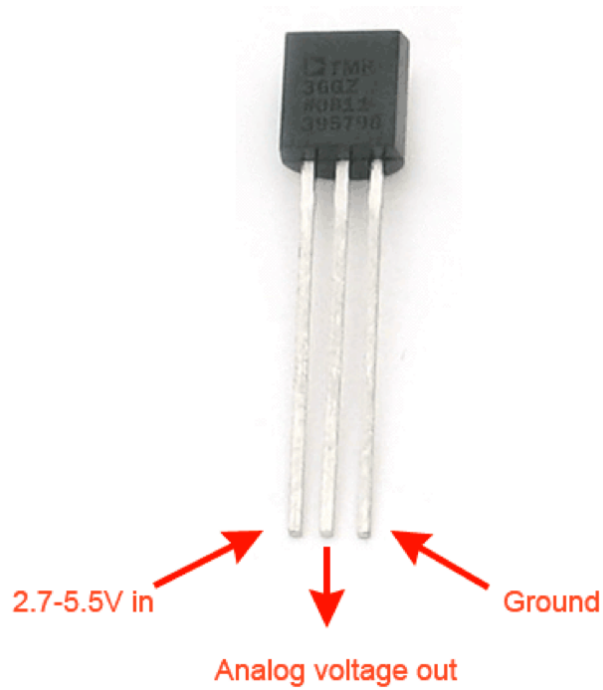


Figure 38.7: TMP36 Temperature Sensor

$$(x_2, y_2) = (1V, 50^\circ) \quad (38.2)$$

From the Datasheet (Figure 38.9) we see that there is a linear relationship between Voltage and degrees Celsius (38.3):

$$y = ax + b \quad (38.3)$$

We can find a and b using the following known formula (38.4):

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \quad (38.4)$$

By putting (38.1) and (38.2) into (38.4) we get:

$$y - 25 = \frac{50 - 25}{1 - 0.75}(x - 0.75) \quad (38.5)$$

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

$$y = 100x - 50 \quad (38.6)$$

Example 38.5.1. Read TMP36 Temperature Data

Python code:

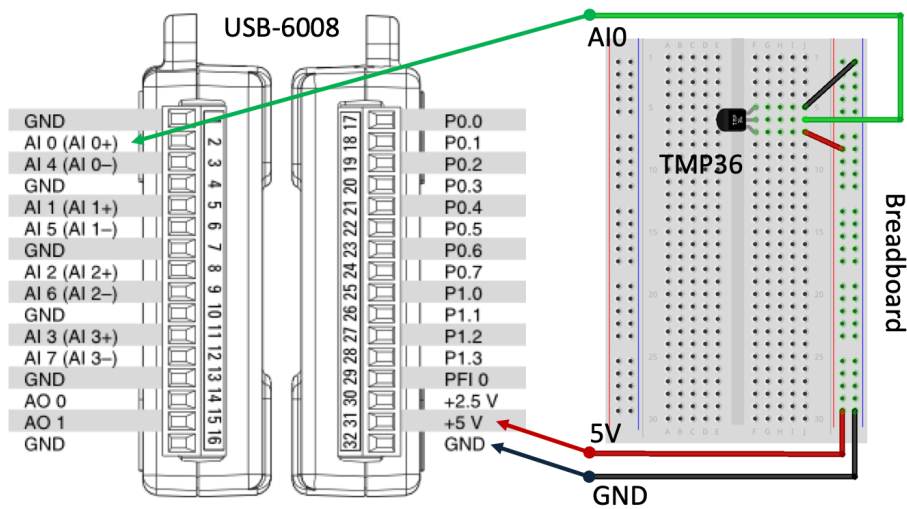


Figure 38.8: TMP36 tmp36 Wiring

```

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         voltage = task.read()
15
16         degreesC = 100*voltage - 50
17
18         print("Sample:", i)
19         print("Voltage Value:", round(voltage,2))
20         print("Celsius Value:", round(degreesC,1))
21         print("\n")
22         time.sleep(1)
23         i = i+1
24
25     task.stop

```

Listing 38.9: 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 38.5.2. Real-Time Plotting of Temperature Data

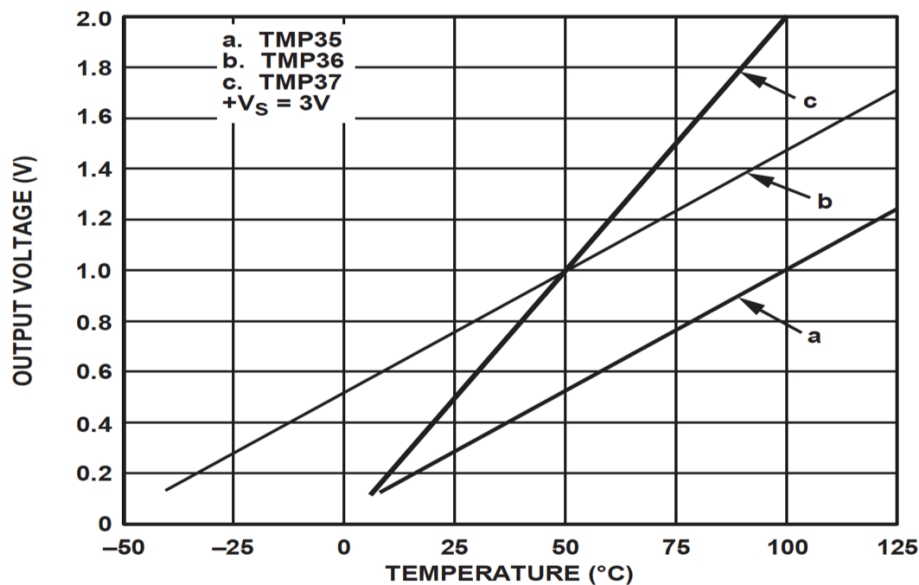


Figure 38.9: TMP3x Datasheet

In this example we will plot the data from the sensor using the Real-time plotting examples shown in Chapter 37.

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)
57 plt.show()
58 task.stop

```

Listing 38.10: Real-Time Plotting of Temperature Data

[End of Example]

38.5.2 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 38.5.3. 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 an non-linear relationship between resistance and excitement. To find the temperature we can use the following equation (Steinhart-Hart equation):

$$\frac{1}{T_K} = A + B \ln(R) + C(\ln(R))^3 \quad (38.7)$$

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:

$$T_K = \frac{1}{A + B \ln(R) + C(\ln(R))^3} \quad (38.8)$$

The Temperature in degrees Celsius will then be:

$$T_C = T_K - 273.15 \quad (38.9)$$

Wiring...

Figure 38.10 shows how we wire the components together.

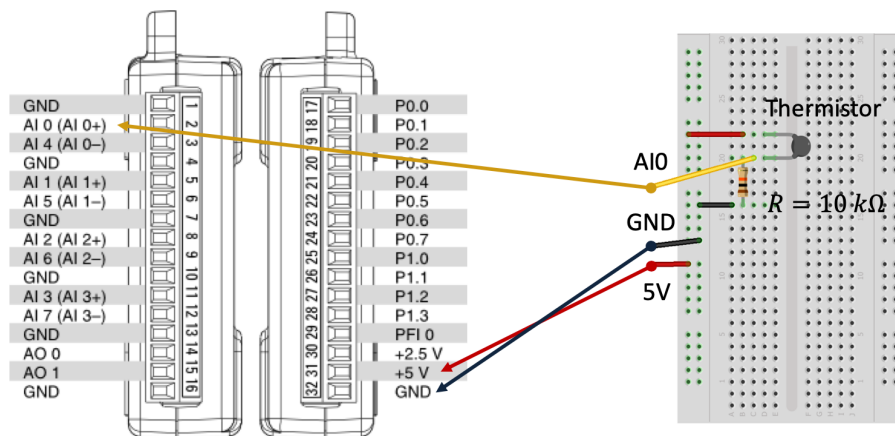


Figure 38.10: Thermistor Wiring

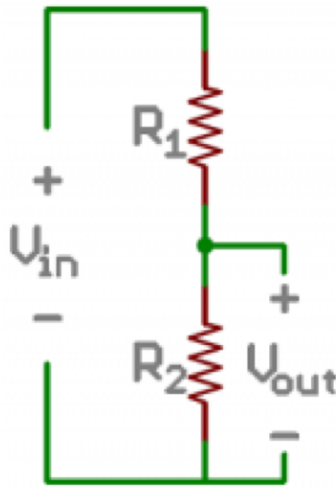


Figure 38.11: Voltage Divider

The wiring is called a "Voltage divider".

Figure 38.11 shows a general Voltage Divider.

A general Voltage Divider has the following equation:

$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2} \quad (38.10)$$

The Voltage Divider for our system becomes as shown in Figure 38.12.

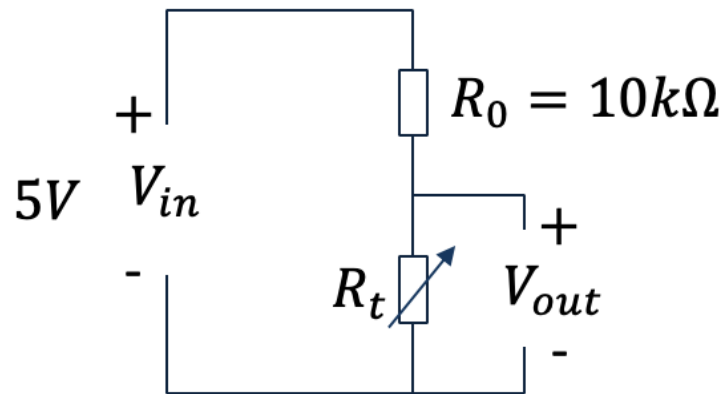


Figure 38.12: Voltage Divider for our System

We then have the following equation:

$$V_{out} = V_{in} \frac{R_t}{R_0 + R_t} \quad (38.11)$$

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:

$$R_t = \frac{V_{out}R_0}{V_{in} - V_{out}} \quad (38.12)$$

We have no ready to start making the Python program for this example.

The program includes 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
24         TempC=25

```

```

24     # Steinhart constants
25     A = 0.001129148
26     B = 0.000234125
27     C = 0.0000000876741
28
29     # Steinhart-Hart Equation
30     TempK = 1 / (A + (B * np.log(Rt)) + (C * pow(np.log(Rt),3)))
31
32     # Convert from Kelvin to Celsius
33     TempC = TempK - 273.15
34
35     print("Sample:", i)
36     print("Voltage Value:", round(Vout,2))
37     print("Celsius Value:", round(TempC,1))
38     print("\n")
39     time.sleep(1)
40     i = i+1
41
42     task.stop

```

Listing 38.11: Read Thermistor Temperature Data

[End of Example]

Example 38.5.4. Real-Time Plotting of Thermistor Temperature Data

Python code:

1 See previous examples

Listing 38.12: Real-Time Plotting of Thermistor Temperature Data

[End of Example]

38.5.3 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 38.13 shows the TC-01 Thermocouple Device from NI.

Example 38.5.5. Real-Time Plotting of Thermistor Temperature Data

Python code:



Figure 38.13: TC-01 Thermocouple Device

```
1 import nidaqmx
2
3 task = nidaqmx.Task()
4
5 task.ai_channels.add_ai_thrmcp1_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 38.13: 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]

38.6 Data Logging

Python has several functions for creating, reading, updating, and deleting files.

Part VIII

Python Database
Development

Chapter 39

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 39.1 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/>

39.1 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

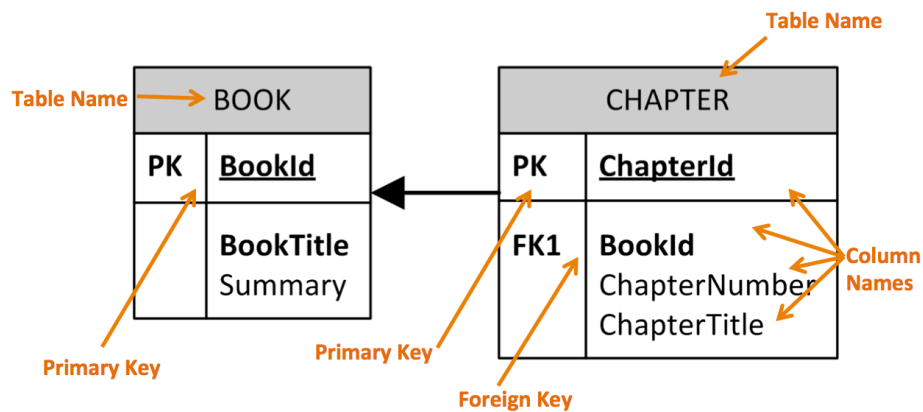


Figure 39.1: ER Diagram Example

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/>

39.2 SQL Server

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

39.3 MySQL

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

39.4 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 40

MongoDB with Python

Here we will learn how we can use Python for communication with MongoDB. We will learn how we connect to a database, how we can insert data into the database and retrieve data from the database.

40.1 Introduction to 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>

40.2 MongoDB with Python

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

Python needs a MongoDB driver to access the MongoDB database. Many different drivers do exist, so you just need to choose one.

40.2.1 PyMongo

The PyMongo distribution contains tools for interacting with MongoDB database from Python

<https://pypi.org/project/pymongo/>

Installation is done using PIP:

```
1 python -m pip install pymongo
```

40.3 Additional Resources

Tutorials:

https://www.w3schools.com/python/python_mongodb_getstarted.asp

Part IX

Python Application Development

Chapter 41

Development of Applications with Python

Python is popular within computation, but can be used within many other applications and can be integrated and used in combination with other programming languages, e.g., for development of Web Applications, etc.

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.

Python can be used for creating Web pages (in combination with HTML, CSS, JavaScript). Python is then typically used on the server-side, while HTML, CSS and JavaScript are used on the client-side.

An example is Django, which is a server-side Python framework used for creating dynamic web pages.

Python can also be used for programming and creating Raspberry Pi applications.

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:

- 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
- Databases (such as SQL Server and MySQL) and using the Structured Query Language (SQL)
- 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.

A good resource or starting point for creating Applications for Python is:

Applications for Python [18]:

<https://www.python.org/about/apps/>

41.1 Mathematics, Science and Engineering

Python is probably best suited for "ad-hoc" numerical calculations, analysis, simulations, etc. These kinds of applications has also been the main focus in this textbook. Other kinds of applications will briefly be covered in this chapter. More details will be covered in later chapters and in other textbooks in my Python textbooks series, which you can read more of in the Preface of this textbook.

The SciPy is a collection of packages for mathematics, science, and engineering, which has been thoroughly reviewed earlier in this textbook.

41.2 Desktop GUI Applications

Python don't come with builtin tools for creating traditional desktop GUI applications, so you need to use an external GUI packages for this purpose.

In my opinion, tools like Visual Studio where you can create professional GUI applications using either the C or VB.NET languages in one integrated packages is a better choice.

Another good alternative is LabVIEW, which has powerful GUI features in combination with extensive hardware integration.

Other tools (Integrated Programming Environments, IDE) and programming languages for GUI applications os Xcode on macOS, which can be used for creating macOS desktop applications and apps for iPhone and iPad.

For Android development you have Android Studio. Here you can use programming languages like Java and Kotlin.

Since this is a Python textbook, let's go back to the options we have if we want to create desktop GUI applications with Python.

Python has different desktop GUI frameworks like:

- PyQt
- Tkinter
- WxPython
- PyGUI
- PySide2
- Kivy

These are just some of the options we have, for a comprehensive overview of GUI frameworks for Python see the following:

<https://docs.python.org/3/library/othergui.html#other-gui-packages>
<https://wiki.python.org/moin/GuiProgramming>

PyQt and wxPython, all have a modern look and feel and more widgets than Tkinter.

This is also a bit of a problem when it comes to desktop GUI development with Python. You have so many choices, and sometimes it's better with one good option than many half good options.

41.2.1 PyQt

PyQt brings together the Qt C++ cross-platform application framework and the cross-platform interpreted language Python. Qt is a cross-platform GUI toolkit.

Qt also includes Qt Designer, a graphical user interface designer. PyQt is able to generate Python code from Qt Designer. It is also possible to add new GUI controls written in Python to Qt Designer.

For more information about PyQt:

<https://riverbankcomputing.com/software/pyqt/intro>

For more information about Qt:

<https://www.qt.io>

PyQt Tutorials:

<https://likegeeks.com/pyqt5-tutorial/>

<https://build-system.fman.io/pyqt5-tutorial>

41.2.2 PySide2

PySide2 is the official Python module from the Qt for Python project, which provides access to the complete Qt framework.

The originally PySide framework was originally released by Nokia, then owner of Qt. After Nokia sold Qt in 2011, PySide was no longer maintained. Then PySide2 was established and maintained by a community. Finally, in 2016, the Qt company committed to officially support the PySide2 project. So basically, PySide2 is very similar to PyQt.

The downside with PySide2 (August 2019) is that it is still in "beta" (Technical Preview).

For more information about PySide2:

<https://pypi.org/project/PySide2/>

https://wiki.qt.io/Qt_for_Python

41.2.3 Tkinter

Another popular GUI framework is Tkinter.

For more information about Tkinter:

<https://docs.python.org/2/library/tkinter.html>

41.2.4 WxPython

WxPython is a cross-platform GUI toolkit for the Python language.

For more information about WxPython:

<https://www.wxpython.org>

<https://wiki.wxpython.org/Getting>

41.3 Web Applications

Python can be used for creating Web pages (in combination with HTML, CSS, JavaScript). Python is then typically used on the server-side, while HTML, CSS and JavaScript are used on the client-side.

An example is Django, which is a server-side Python framework used for creating dynamic web pages.

Read more about Django here:

<https://www.djangoproject.com>

Other popular web application frameworks and programming languages are ASP.NET and PHP.

You may read more about web programming in general here:

<https://www.halvorsen.blog/documents/programming/web/>

41.4 Database Applications

Python can be used for communication with a database system such as SQL Server or MySQL. Python has different packages for communication with different database systems, both SQL databases (e.g., SQL Server, MySQL, MariaDB, etc.) and so-called NoSQL databases (e.g., MongoDB).

Here you can learn more about Database Systems and SQL, download examples and get additional resources, see videos, etc.:

<https://www.halvorsen.blog/documents/technology/database/>

41.4.1 SQL Server

SQL Server is a Database System from Microsoft. SQL Server comes in different editions, for basic, personal use SQL Server Express is recommended because it is simple to use and it is free.

Read more about SQL Server here:

https://www.halvorsen.blog/documents/technology/database/sql_server.php

41.4.2 MySQL

MySQL is an open-source and widely used Relational Database Management System (RDBMS).

MySQL comes in different editions, both paid (Enterprise) and free versions (Community).

In addition to the Database itself, the MySQL Workbench is nice to have. MySQL Workbench visual database design tool, used for Database Modelling, etc. Another handy tool is phpMyAdmin. phpMyAdmin is a free software tool written in PHP, intended to handle the administration of MySQL.

Read more about MySQL here:

<https://www.halvorsen.blog/documents/technology/database/mysql.php>

41.4.3 MariaDB

MariaDB is a spinoff from the more famous MySQL Database System.

The MariaDB database server is published as free and open source software. MariaDB has compatibility with MySQL in most situations. MariaDB is said to have slightly better performance than MySQL.

Read more about MariaDB here:

<https://www.halvorsen.blog/documents/technology/database/mariadb.php>

41.4.4 MongoDB

MongoDB is a general purpose, document-based, distributed database.

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program.

You can download MongoDB from:

<https://www.mongodb.com>

Chapter 42

Python Integration with 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 Visual Studio 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

Here you can learn more about Visual studio and C, download examples and get additional resources, see videos, etc.:

<https://www.halvorsen.blog/documents/programming/csharp/>

For an introduction to the Python integration in Visual Studio, see Chapter 19.

Chapter 43

Python Integration with LabVIEW

43.1 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.

43.2 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/

Use the Python functions to call Python code from LabVIEW. See Figure 43.1

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

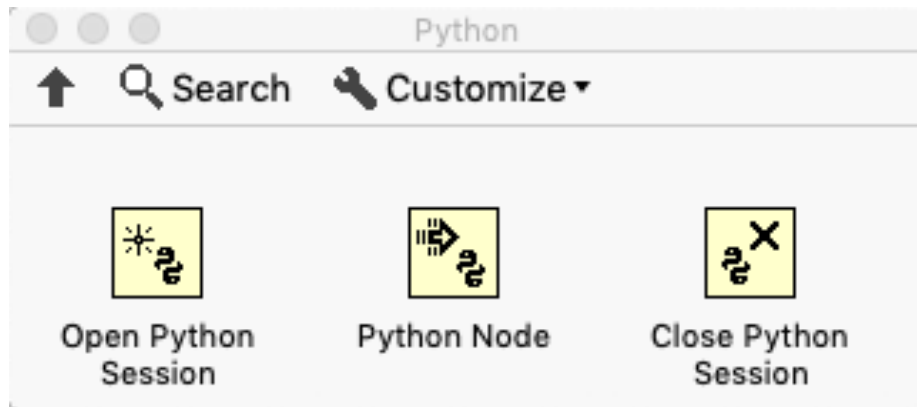


Figure 43.1: Python Integration in LabVIEW

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 43.2.1. 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:

$$T_f = (T_c \times 9/5) + 32 \quad (43.1)$$

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \quad (43.2)$$

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 43.1: 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 43.2: 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 43.2 we see the Front Panel.

In Figure 43.3 we see the Block Diagram.

In Figure 43.4 we see LabVIEW Code for calling both Python functions (c2f and f2c) from LabVIEW.

[End of Example]

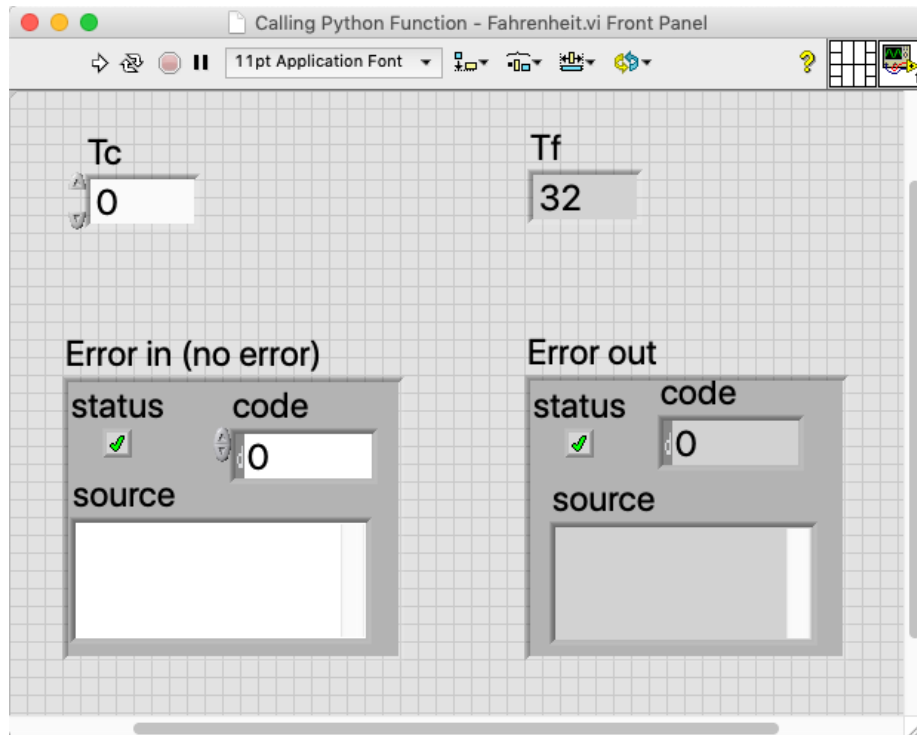


Figure 43.2: Python Integration in LabVIEW

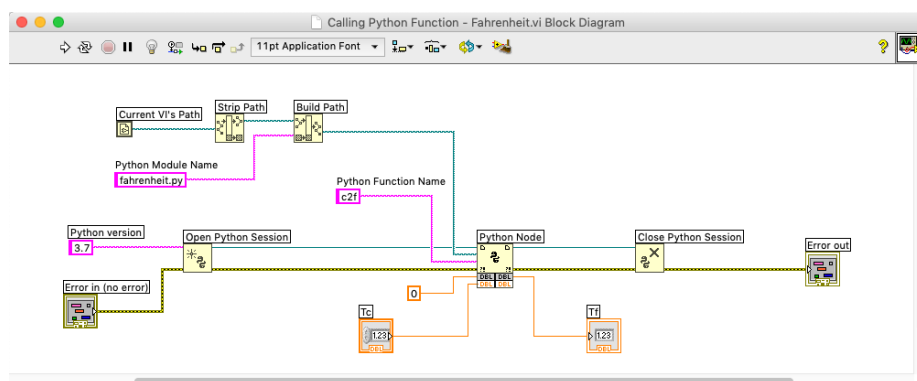


Figure 43.3: Python Integration in LabVIEW

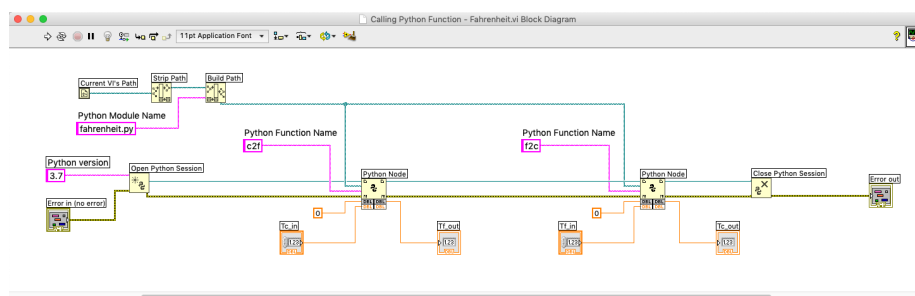


Figure 43.4: Python Integration in LabVIEW

Chapter 44

Raspberry Pi and Python

44.1 What is Raspberry Pi?

The Raspberry Pi is a credit-card-sized computer that plugs into your TV and a keyboard. It is a capable little computer which can be used in electronics projects, and for many of the things that your desktop PC does.

Raspberry Pi is very popular in IoT projects and applications.

For more information and resources regarding Raspberry Pi:

https://www.halvorsen.blog/documents/technology/iot/raspberry_pi.php

Other Resources:

<https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/programming-in-python>

First, before you start programming Python on a Raspberry Pi device, you need to install an operating system like Raspbian. Raspbian is a Linux distribution tailor made for Raspberry Pi.

Raspbian comes also pre-installed Python.

For more information about Raspbian:

<https://www.raspberrypi.org/downloads/raspbian/>

Chapter 45

Machine Learning with Python

45.1 Introduction to Machine Learning

Here you can learn more about Machine Learning, download examples and get additional resources, see videos, etc.:

https://www.halvorsen.blog/documents/technology/machine_learning/

Part X

Resources

Chapter 46

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 46.1 shows the book covers.

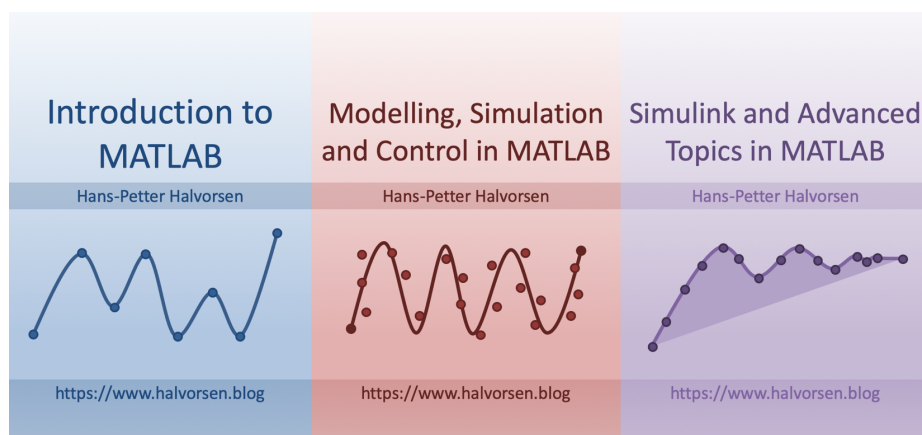


Figure 46.1: MATLAB Textbooks

46.1 Use Python inside MATLAB

To call Python libraries from MATLAB, you need to install the reference implementation for Python (CPython).

For more information, see the following:

<https://se.mathworks.com/help/matlab/getting-started-with-python.html>

MATLAB and Python must also be the same architecture/bit version. If you have MATLAB 64 bits version, you need Python 64 bits. If you have MATLAB 32 bits version, you need Python 32 bits.

In MATLAB start by finding information about the default Python Environment used by MATLAB:

```
1 >> pyversion
```

This gives the following on my Windows10 computer:

```
1 > pyversion
2
3     version: '3.7 '
4     executable: 'C:\Users\hansha\AppData\Local\Programs\Python\
Python37-64\pythonw.exe '
5     library: 'C:\Users\hansha\AppData\Local\Programs\Python\
Python37-64\python37.dll '
6     home: 'C:\Users\hansha\AppData\Local\Programs\Python\
Python37-64 '
7     isloaded: 1
```

If MATLAB and Python don't the same architecture/bit version, you need to either install the proper version of either MATLAB or Python. You may also need to "Add Python to Path" as described in Chapter 3.

Example 46.1.1. 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:

$$T_f = (T_c \times 9/5) + 32 \quad (46.1)$$

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \quad (46.2)$$

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 46.1: 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 46.2: 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 46.3: Calling Python Library and Functions from MATLAB

[End of Example]

46.2 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 47

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 [19]:
<https://docs.python.org/3/library/index.html>

47.1 Python Distributions

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

47.2 Python Libraries

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

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

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

47.3 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>

47.4 Python Tutorials

Python Tutorial - w3schools.com [13]:
<https://www.w3schools.com/python/>

The Python Guru [20]:
<https://thepythonguru.com>

Wikibooks - A Beginner's Python Tutorial:
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/>

47.5 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] Jupyter, “Jupyter - <https://jupyter.org/>,” 2018.
- [17] JupyterHub, “Jupyterhub - <http://jupyter.org/hub>,” 2018.

- [18] python.org, “Applications for python - <https://www.python.org/about/apps/>,” 2018.
- [19] python.org, “The python standard library - <https://docs.python.org/3/library/>,” 2018.
- [20] T. P. Guru, “The python guru - <https://thepythonguru.com>,” 2018.

Part XI

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.

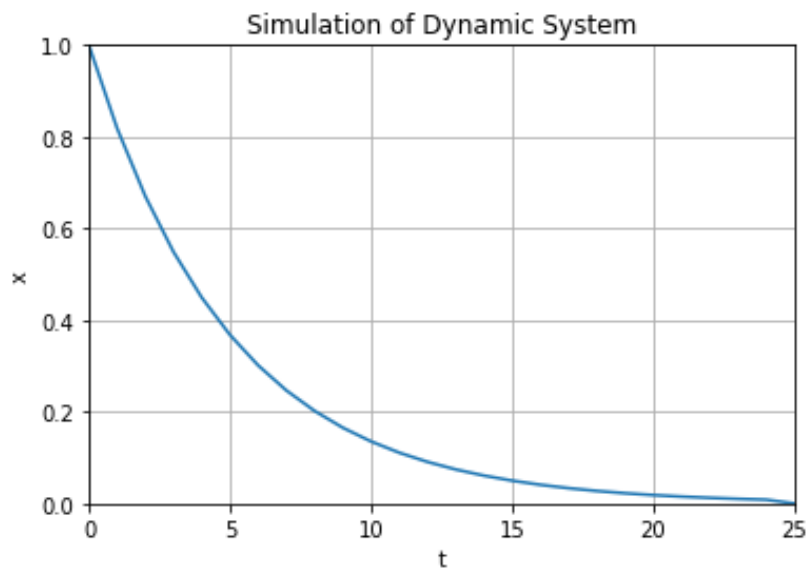


Figure 1: Simulation of Dynamic System

[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 calcexpression(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.calcexpression(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 $p=0.5$ 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
18 # Simulation
19 for k in range(N):
20     xk1 = xk + Ts* (b * xk - p * xk**2);
21     xk = xk1
22     data.append(xk1)
23
24 # Plot the Simulation Results
25 t = np.arange(0, Tstop+Ts, Ts)
26
27 plt.plot(t, data)
28 plt.title('Simulation of Bacteria Population')
29 plt.xlabel('t [s]')
30 plt.ylabel('x')
31 plt.grid()
32 plt.axis([0, 1, 0, 100])
33 plt.show()

```

The simulation gives the results as shown in Figure 2.

[End of Solution]

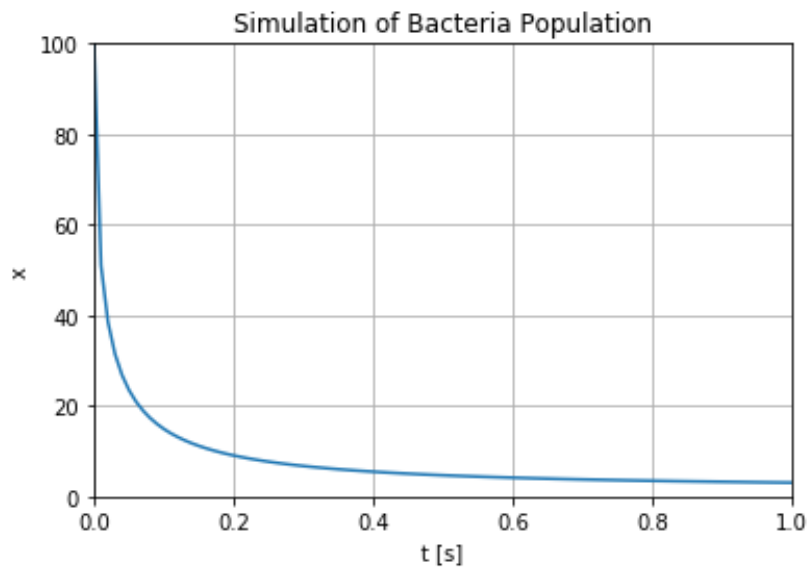


Figure 2: Simulation of Bacteria Population

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.

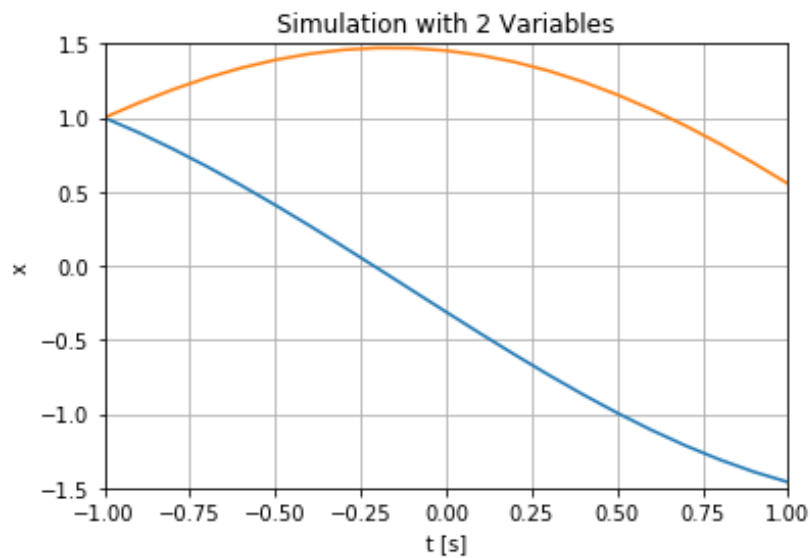


Figure 3: Simulation Example with 2 Variables

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 Science and Engineering