

MAKING LIBRETEXTS INTERACTIVE WITH BINDER: A PRACTICAL GUIDE



Carl Greco

Making LibreTexts Interactive with Binder: A Practical Guide

This text is disseminated via the Open Education Resource (OER) LibreTexts Project (<https://LibreTexts.org>) and like the thousands of other texts available within this powerful platform, it is freely available for reading, printing, and "consuming."

The LibreTexts mission is to bring together students, faculty, and scholars in a collaborative effort to provide an accessible, and comprehensive platform that empowers our community to develop, curate, adapt, and adopt openly licensed resources and technologies; through these efforts we can reduce the financial burden borne from traditional educational resource costs, ensuring education is more accessible for students and communities worldwide.

Most, but not all, pages in the library have licenses that may allow individuals to make changes, save, and print this book. Carefully consult the applicable license(s) before pursuing such effects. Instructors can adopt existing LibreTexts texts or Remix them to quickly build course-specific resources to meet the needs of their students. Unlike traditional textbooks, LibreTexts' web based origins allow powerful integration of advanced features and new technologies to support learning.



LibreTexts is the adaptable, user-friendly non-profit open education resource platform that educators trust for creating, customizing, and sharing accessible, interactive textbooks, adaptive homework, and ancillary materials. We collaborate with individuals and organizations to champion open education initiatives, support institutional publishing programs, drive curriculum development projects, and more.

The LibreTexts libraries are Powered by [NICE CXone Expert](#) and were supported by the Department of Education Open Textbook Pilot Project, the California Education Learning Lab, the UC Davis Office of the Provost, the UC Davis Library, the California State University Affordable Learning Solutions Program, and Merlot. This material is based upon work supported by the National Science Foundation under Grant No. 1246120, 1525057, and 1413739.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation nor the US Department of Education.

Have questions or comments? For information about adoptions or adaptations contact info@LibreTexts.org or visit our main website at <https://LibreTexts.org>.

The LibreTexts name, logo, and book covers are trademarked by LibreTexts, Inc. (a 501(c)(3) not-for-profit organization) and protected against unauthorized use.

This text was compiled on 09/10/2026

TABLE OF CONTENTS

Licensing

1: Introduction

2: Python Binder Examples

2.1: Python Examples

2.2: Introduction to Digital Signal Processing

Index

Glossary

Detailed Licensing

Detailed Licensing

Licensing

A detailed breakdown of this resource's licensing can be found in **Back Matter/Detailed Licensing**.

This page titled [Licensing](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

1: Introduction

Welcome to a new era of open education, where textbooks are no longer static blocks of text and passive images, but live, executable code sandboxes. *Making LibreTexts Interactive with Binder* bridges the gap between foundational theory and hands-on computational practice. By embedding live Jupyter notebooks directly into open-access pages via Binder, this book transforms the traditional reading experience into an interactive laboratory. Whether you are an educator looking to seamlessly integrate Python, R, or Julia workflows into your curriculum, or a student eager to tweak parameters and visualize data in real time, the examples and frameworks within this guide provide the blueprints for building a truly dynamic, accessible, and resilient digital classroom.

2: PYTHON BINDER EXAMPLES

2.1: Python Examples

2.2: Introduction to Digital Signal Processing

This page titled [2: Python Binder Examples](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

2.1: Python Examples

Technical Guide: Reading a Pre-Installed Data File

When working with data science or statistics modules in LibreTexts, you often need to import datasets stored directly within the platform. To ensure your code (such as Python or R) reads the file smoothly without HTML clutter, you must bypass the standard LibreTexts web viewer and target the raw file directly.

LibreTexts stores uploaded media and CSV files with permanent URLs. However, standard URLs are designed to display the file within the LibreTexts user interface and not for processing within a programming language application. To load the raw data into a programming environment, you must append `?direct=true` to the very end of the URL. This parameter forces the server to serve the raw text rather than a rendered webpage.

```
import pandas as pd
import matplotlib.pyplot as plt

# Read data from CSV file
# Assumes the CSV has no header row
# Read from Assets with "?direct=true" appended to permanent link
filename = "https://commons.libretexts.org/permalink/u790oZqbi6/1adc20f3-f3d0-4887-8ddd-cc"
df = pd.read_csv(filename, header=None)
data = df[0]

# Calculate statistics
data_mean = data.mean()
data_std = data.std()

# Print results
print(f"Data Length: {len(data)}")
print(f"Mean: {data_mean:.2f}")
print(f"Standard Deviation: {data_std:.2f}")

# Create histogram
plt.hist(data, bins='auto', edgecolor='black')
plt.title('Histogram of Data', fontsize=12)
plt.xlabel('Value', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.grid()
plt.show()
```

Installing Missing Libraries in Jupyter Environments

When working in an interactive environment like MyBinder, Jupyter Notebooks, or Google Colab, certain specialized Python libraries may not be pre-installed in the default environment. You can dynamically install these missing packages directly inside your MyBinder segment within the LibreTexts document using shell commands.

To execute a system-level command line instruction from inside a Python cell, precede the command with an exclamation point (often called the **bang operator**, `!`). This tells binder to run the instruction in the underlying operating system shell rather than as Python code.

For example in the MyBinder environment you may execute a command-line (shell) command using the bang operator, `!`, preceding the command to install the scikit-learn library. Use one of the following commands in the script:

Option A: Using pip (Recommended)

The Python package installer ([pip](#)) is typically faster and highly efficient for standard packages inside temporary cloud containers.

```
!pip install scikit-learn
```

Option B: Using Conda

If your environment requires package and dependency management via Anaconda, you can pull the package from the community-driven [conda-forge](#) channel.

```
!conda install -c conda-forge scikit-learn
```

```
!pip install scikit-learn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# 1. Create dummy data (100 samples, 1 feature)
X = np.random.rand(100, 1) * 10
b = 5.0    # intercept
a = 2.5    # slope
y = a * X + b + np.random.randn(100, 1) # y = 2.5X + b + noise
print(f"Feature shape: {X.shape}, Target shape: {y.shape}")

# 2. Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 3. Initialize and fit the model
model = LinearRegression()
model.fit(X_train, y_train)

# 4. Access model parameters
print(f"Slope (Coefficient): {model.coef_[0][0]:.4f}") # Targets ~2.5
print(f"Intercept: {model.intercept_[0]:.4f}") # Targets ~0.0

# 5. Make predictions
predictions = model.predict(X_test)

# 6. Visualize the results
plt.figure(figsize=(8, 4))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, predictions, color='red', linewidth=2, label='Predicted')
plt.xlim(left=0)
plt.ylim(bottom=0)
plt.xlabel('Feature')
plt.ylabel('Target')
plt.title(f'Linear Regression, Slope: {model.coef_[0][0]:.4f}, Intercept: {model.intercept_[0]:.4f}')
plt.legend()
plt.grid(True)
plt.show()
```

Pedagogical Guide: Implementing Formative Assessment via Built-In Exercise Boxes

Formative assessment is a crucial tool for checking student understanding in real time, allowing both the instructor and the learner to identify gaps before moving on to higher-stakes summative evaluations. In interactive textbooks and platforms like LibreTexts, the **built-in exercise box** serves as an immediate, low-stakes checkpoint.

Placing an exercise box directly beneath a newly introduced concept creates an active learning loop. Instead of passively reading through dense material, students must immediately retrieve and apply the information. Because these activities are formative, they are typically graded on completion or not graded at all, reducing anxiety and encouraging experimentation.

Python Exercise 2.1.1

Create a Python script to find the reaction force resulting from two forces in two dimensional space. Forces, F1 and F2, are entered as numpy arrays with x and y coordinates $[F_x, F_y]$. The program should display the total summative force and its reaction force as two dimensional vectors and in polar format as magnitude and angle.

Answer: Applied and Reaction forces $F_{Total} = [-1.0, 5.0] = (5.10, \angle 101.31^\circ)$, $F_{Reaction} = [1.0, -5.0] = (5.10, \angle -78.69^\circ)$

```
import numpy as np

F1 = np.array([1, 2]) # [F1_x, F1_y]
F2 = np.array([-2, 3]) # [F2_x, F2_y]
# Enter your code here
```

Solution available at the end of this section.

Code Block in Solution

- Create an answer as a place holder
- Open editor and switch to HTML mode.
- Replace the place holder with the code between

your code goes here

Answers

```
def vector_polar(vector):
    """Function to calculate magnitude and angle in degrees of a vector"""
    magnitude = np.linalg.norm(vector)
    angle = np.arctan2(vector[1], vector[0]) * (180 / np.pi)
    return magnitude, angle

F_total = F1 + F2
F_mag, F_angle = vector_polar(F_total)

print(f"Total Force Vector: ({F_total[0]:.2f}, {F_total[1]:.2f})")
print(f"Magnitude of Total Force: {F_mag:.2f}")
print(f"Angle of Total Force (degrees): {F_angle:.2f}")

F_Reaction = -F_total
F_Reaction_mag, F_Reaction_angle = vector_polar(F_Reaction)

print(f"\nReaction Force Vector: ({F_Reaction[0]:.2f}, {F_Reaction[1]:.2f})")
print(f"Magnitude of Reaction Force: {F_Reaction_mag:.2f}")
print(f"Angle of Reaction Force (degrees): {F_Reaction_angle:.2f}")
```


2.2: Introduction to Digital Signal Processing

Learning Objectives

- Segue for DSP chapter.

Not only do we have analog signals --- signals that are real- or complex-valued functions of a continuous variable such as time or space --- we can define **discrete/digital** ones as well. Digital signals are **sequences**, functions defined only for the integers. We thus use the notation $s(n)$ to denote a discrete-time one-dimensional signal such as a digital music recording and $s(m,n)$ for a discrete-"time" two-dimensional signal like a photo taken with a digital camera. Sequences are fundamentally different than continuous-time signals. For example, continuity has no meaning for sequences.

Despite such fundamental differences, the theory underlying digital signal processing mirrors that of analog signals: Fourier transforms, linear filtering and linear systems parallel what previous chapters described. These similarities make it easy to understand the definitions and why we need them, but the similarities should not be construed as "analog wannabes." We will discover that digital signal processing is **not** an approximation to analog processing. We must explicitly worry about the fidelity of converting analog signals into digital ones. The music stored on CDs, the speech sent over digital cellular telephones, and the video carried by digital television all evidence that analog signals can be accurately converted to digital ones and back again.

The key reason why digital signal processing systems have a technological advantage today is the **computer**: computations, like the Fourier transform, can be performed quickly enough to be calculated as the signal is produced, and programmability means that the signal processing system can be easily changed. This flexibility has obvious appeal and has been widely accepted in the marketplace. Programmability means that we can perform signal processing operations impossible with analog systems (circuits). We will also discover that digital systems enjoy an **algorithmic** advantage that contributes to rapid processing speeds: Computations can be restructured in non-obvious ways to speed the processing. This flexibility comes at a price, a consequence of how computers work. How do computers perform signal processing?

Note

Taking a systems viewpoint for the moment, a system that produces its output as rapidly as the input arises is said to be a **real-time** system. All analog systems operate in real-time; digital ones that depend on a computer to perform system computations may or may not work in real-time. Clearly, we need real-time signal processing systems. Only recently have computers become fast enough to meet real-time requirements while performing non-trivial signal processing.

Digital signal processing has been applied to many different areas including medicine, telecommunication, military, and scientific to mention a few. For example, the number of sun spots have been observed since the mid 1700's and analyzed to determine patterns of the frequency of their occurrence.

Example

The following is an example of digital signal processing applied to scientific data analysis. The sunspot data was obtained from the [Solar Influences Data Analysis Center \(SILSO\)](#) (opens in new window)CC BY-NC 4.0. Also, see the [National Solar Observatory](#) (NSO) for more information solar activity.

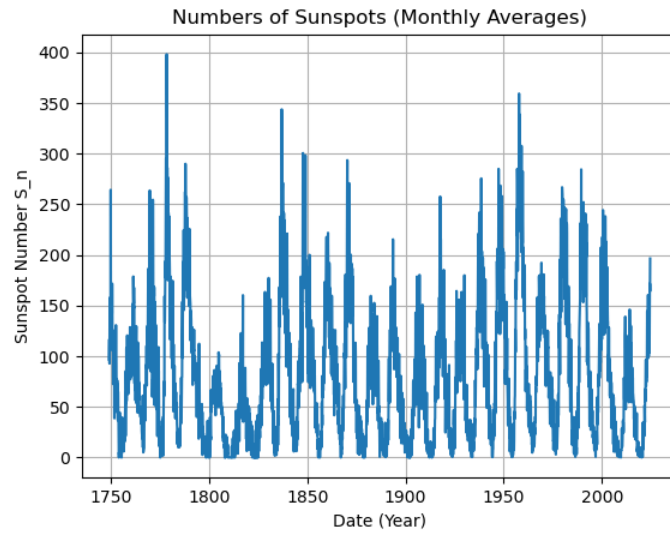


Figure 2.2.1: Sunspot Monthly Average Data. Late 1700's to early 1800's is know as the Dalton Minimum.

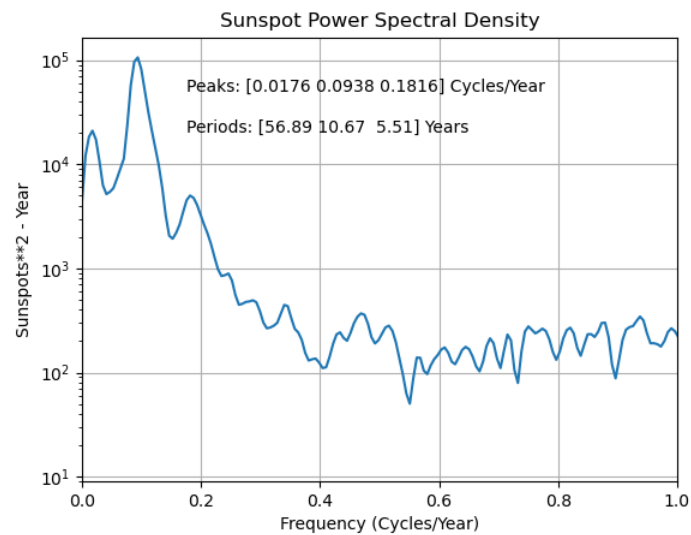


Figure 2.2.2: Power Spectral Density of Sunspot Data.

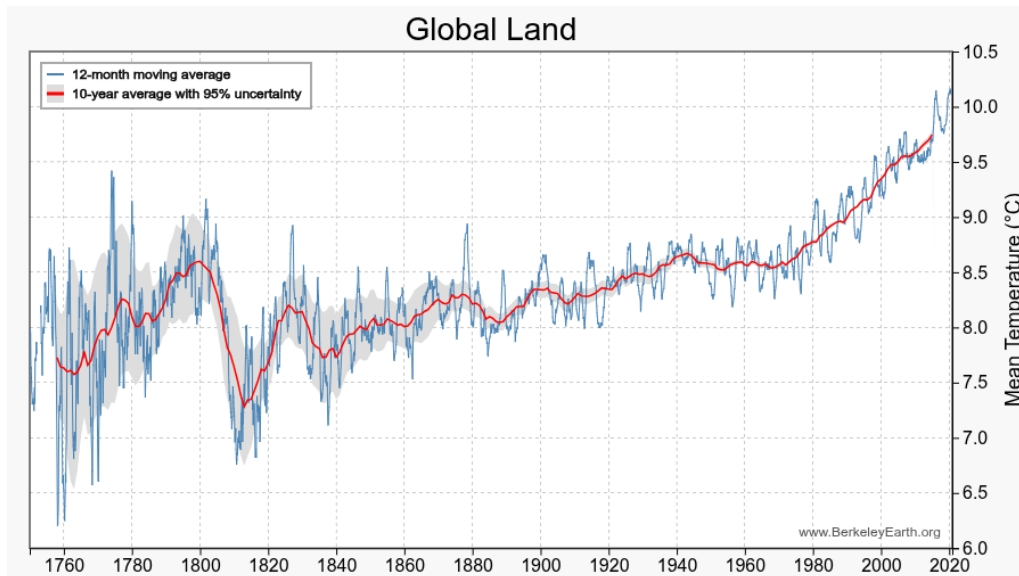


Figure 2.2.3: Global average land temperature. (Berkeley Earth [Global temperature history](https://berkeleypath.org/) [berkeleyearth.org])

The average number of sunspots that appeared on a monthly basis is displayed in the above graph, and the frequency analysis of these sunspots was obtained from power spectral density (PSD) analysis of this sunspot data as shown in the second graph. Not surprisingly, there was a peak in the PSD analysis at 0.0938 cycles/year for a period of 10.67 years. This corresponds to the observed cyclic variation of sunspots in the above graph. The PSD analysis also reveals that there are two additional peaks: one at 0.0176 cycles/year with a period of 56.89 years and another at 0.1816 cycles/year for a 5.51 year period. The ~57 year variation may be observed as the waxing and waning in the peak heights that occur every ~10.7 years corresponding to the completion of the sun's magnetic pole reversal. The higher frequency peak that occurs at ~5.5 year cycle, is not noticeable in the sunspot data in the above figure which is consistent with its diminished peak magnitude in the PSD analysis relative to the other two peaks.

Python Code

```
'''
Sunspot Number - Power Spectral Density
Download Sunspot Data and Estimate Power Spectral Density
Solar Influences Data Analysis Center (WDC-SILSO) [https://www.sidc.be/SILSO/home]
Royal Observatory of Belgium, Brussels
% Input data Monthly mean total sunspot number (1/1749 - 8/2024)
Columns separated by one or more spaces
    Column 1-2: Gregorian calendar date
        - Year
        - Month
    Column 3: Date in fraction of year.
    Column 4: Monthly mean total sunspot number.
    Column 5: Monthly mean standard deviation of the input sunspot numbers.
    Column 6: Number of observations used to compute the monthly mean total
              sunspot number.
    Column 7: Definitive/provisional marker. A "1" indicates that the value is
              definitive. A '0' indicates that the value is still provisional.
'''

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
import sys

def load_csv_with_fallback(primary_path, fallback_path, **kwargs):
    """Attempts to read a CSV file from a primary location.
```

Falls back to a secondary location if the primary fails.

Parameters:

- `primary_path` (str): The main file path or URL to try first.
- `fallback_path` (str): The backup file path or URL.
- `kwargs`: Additional arguments passed directly to `pd.read_csv()`.

Returns:

- `pd.DataFrame`: The loaded DataFrame if successful.

Raises:

- `FileNotFoundError`: If both locations fail to load.

```
"""
# 1. Try the primary location
try:
    print(f"Attempting to read primary file: {primary_path}")
    df = pd.read_csv(primary_path, **kwargs)
    print("Successfully loaded from primary location.")
    return df

except (FileNotFoundError, Exception) as e:
    print(f"Primary location failed. Reason: {e}")
    print("-" * 40)

    # 2. Try the fallback location
    try:
        print(f"Attempting to read fallback file: {fallback_path}")
        df = pd.read_csv(fallback_path, **kwargs)
        print("Successfully loaded from fallback location.")
        return df

    except (FileNotFoundError, Exception) as fallback_e:
        print(f"Fallback location also failed. Reason: {fallback_e}")
        # Raise an error if both attempts fail
        raise FileNotFoundError(
            f"Critical: Could not load CSV from either \n"
            f"Primary: '{primary_path}' or \n"
            f"Fallback: '{fallback_path}'"
        ) from fallback_e

# Attempt to read the local file first
'''
Read Sunspot Data from local file or online source
- File format: csv or txt
- Columns separated by one or more spaces or semicolon
- Use sep=';' for csv file or sep=r'\s+' for txt file with columns separated by one or
- If the file is not found, it will try to read from the online source
- If both fail, it will print an error message and set df to None
'''

filename_csv = "https://commons.libretexts.org/permalink/IKTzPvY930/13d7b3b3-b897-4f9b-907
filename_url = "https://www.sidc.be/SILSO/DATA/SN_m_tot_V2.0.csv" # Online source
try:
    # You can pass standard read_csv parameters like sep, encoding, etc.
    df = load_csv_with_fallback(filename_csv, filename_url, sep=',|;', index_col=False)
    # Do something with the data
    print("\nData Preview:")
    print(df)

except Exception as final_error:
    print(f"\nScript execution failed: {final_error}")
```

```
# Design the FIR lowpass filter & filter data
# Order of the filter
# Sampling frequency (cycles/year)
fs = 12      # Once per month (12 times per year)
# Cutoff frequency (cycles/year)
fc = 0.4     # Cutoff frequency cycles/year
order = 12
M1 = order + 1
# Lowpass Filter the Data
# Design the FIR lowpass filter using the window method
h1 = signal.firwin(M1, fc, window='hann', fs = fs)
y_filtered = signal.filtfilt(h1,[1],df["Sunspots"])

# Display Graph of Sunspots
plt.figure(figsize=(12,6))
plt.plot(df["Date"],df["Sunspots"], label='Monthly Averages')
plt.plot(df["Date"], y_filtered, 'r-', linewidth=1.5, label=f'Lowpass Filtered, fc = {fc}')
plt.xlim(df["Date"].min(), df["Date"].max())
plt.grid()
plt.xlabel("Date (Year)")
plt.ylabel("Sunspot Number  $S_n$ ")
plt.title("Numbers of Sunspots 1749 to Present (Monthly Averages), Lowpass Filtered with R")
plt.legend()

# Power Spectral Density Estimation
NFFT = 2048
nyears = 100
f, Pxx = signal.welch(df["Sunspots"].to_numpy(),fs=fs,window='hann',nperseg=12*nyears, \
    noverlap=12*nyears//2,nfft=NFFT,detrend='linear',return_onesided=True,scaling='density')

plt.figure(figsize=(12,6))
plt.semilogy(f, Pxx)
plt.xlim(0,1)
plt.grid()
plt.xlabel('Frequency (Cycles/Year)')
plt.ylabel('Sunspots**2 - Year')
plt.title('Sunspot Power Spectral Density')
EndRange = int(0.25*NFFT/12)
peaks, _ = signal.find_peaks(Pxx[0:EndRange])
plt.text(0.2,5e4,rf'Peaks: {np.around(peaks*fs/NFFT, decimals=4)} Cycles/Year',fontsize=16)
plt.text(0.2,2e4,rf'Periods: {np.around(np.reciprocal(peaks*fs/NFFT), decimals=2)} Years',
plt.show()
```

This page titled [2.2: Introduction to Digital Signal Processing](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by Carl Greco.

- [5.1: Introduction to Digital Signal Processing](#) by Don H. Johnson is licensed CC BY 1.0. Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

INDEX

A

analog signals

[2: Python Binder Examples](#)

[2.2: Introduction to Digital Signal Processing](#)

C

computational practice

[1: Introduction](#)

computations

[2: Python Binder Examples](#)

computer technology

[2.2: Introduction to Digital Signal Processing](#)

D

data analysis

[2.2: Introduction to Digital Signal Processing](#)

data reading

[2: Python Binder Examples](#)

data science

[2.1: Python Examples](#)

data visualization

[2.1: Python Examples](#)

digital signal processing

[2: Python Binder Examples](#)

[2.2: Introduction to Digital Signal Processing](#)

discrete signals

[2.2: Introduction to Digital Signal Processing](#)

E

educational technology

[1: Introduction](#)

executable code

[1: Introduction](#)

F

Fourier Transforms

[2.2: Introduction to Digital Signal Processing](#)

I

import datasets

[2.1: Python Examples](#)

interactive exercises

[2: Python Binder Examples](#)

interactive textbooks

[1: Introduction](#)

J

Julia

[1: Introduction](#)

Jupyter

[2.1: Python Examples](#)

Jupyter notebooks

[1: Introduction](#)

L

LibreTexts

[1: Introduction](#)

linear filtering

[2.2: Introduction to Digital Signal Processing](#)

M

medicine

[2: Python Binder Examples](#)

O

open education

[1: Introduction](#)

P

pandas

[2.1: Python Examples](#)

Power spectral density

[2.2: Introduction to Digital Signal Processing](#)

Python

[1: Introduction](#)

[2: Python Binder Examples](#)

[2.1: Python Examples](#)

Python package

[2: Python Binder Examples](#)

R

R

[1: Introduction](#)

[2.1: Python Examples](#)

S

scientific analysis

[2: Python Binder Examples](#)

signal processing

[2.2: Introduction to Digital Signal Processing](#)

statistics

[2.1: Python Examples](#)

T

technical guide

[2.1: Python Examples](#)

telecommunications

[2.2: Introduction to Digital Signal Processing](#)

GLOSSARY

S

Sample Word 1 | Sample Definition 1

This page titled [Glossary](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

DETAILED LICENSING

Overview

Title: [Making LibreTexts Interactive with Binder: A Practical Guide](#)

Webpages: 15

All licenses found:

- [CC BY-SA 4.0](#): 86.7% (13 pages)
- [Undeclared](#): 13.3% (2 pages)

By Page

- [Making LibreTexts Interactive with Binder: A Practical Guide](#) - [CC BY-SA 4.0](#)
 - [Front Matter](#) - [CC BY-SA 4.0](#)
 - [TitlePage](#) - [CC BY-SA 4.0](#)
 - [InfoPage](#) - [CC BY-SA 4.0](#)
 - [Table of Contents](#) - [Undeclared](#)
 - [Licensing](#) - [CC BY-SA 4.0](#)
 - [1: Introduction](#) - [CC BY-SA 4.0](#)
 - [2: Python Binder Examples](#) - [CC BY-SA 4.0](#)
 - [2.1: Python Examples](#) - [CC BY-SA 4.0](#)
 - [2.2: Introduction to Digital Signal Processing](#) - [CC BY-SA 4.0](#)
 - [Back Matter](#) - [CC BY-SA 4.0](#)
 - [Index](#) - [CC BY-SA 4.0](#)
 - [Glossary](#) - [CC BY-SA 4.0](#)
 - [Detailed Licensing](#) - [CC BY-SA 4.0](#)
 - [Detailed Licensing](#) - [Undeclared](#)

This page titled [Making LibreTexts Interactive with Binder: A Practical Guide](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

DETAILED LICENSING

Overview

Title: [Making LibreTexts Interactive with Binder: A Practical Guide](#)

Webpages: 15

All licenses found:

- [CC BY-SA 4.0](#): 86.7% (13 pages)
- [Undeclared](#): 13.3% (2 pages)

By Page

- [Making LibreTexts Interactive with Binder: A Practical Guide](#) - [CC BY-SA 4.0](#)
 - [Front Matter](#) - [CC BY-SA 4.0](#)
 - [TitlePage](#) - [CC BY-SA 4.0](#)
 - [InfoPage](#) - [CC BY-SA 4.0](#)
 - [Table of Contents](#) - [Undeclared](#)
 - [Licensing](#) - [CC BY-SA 4.0](#)
 - [1: Introduction](#) - [CC BY-SA 4.0](#)
 - [2: Python Binder Examples](#) - [CC BY-SA 4.0](#)
 - [2.1: Python Examples](#) - [CC BY-SA 4.0](#)
 - [2.2: Introduction to Digital Signal Processing](#) - [CC BY-SA 4.0](#)
 - [Back Matter](#) - [CC BY-SA 4.0](#)
 - [Index](#) - [CC BY-SA 4.0](#)
 - [Glossary](#) - [CC BY-SA 4.0](#)
 - [Detailed Licensing](#) - [CC BY-SA 4.0](#)
 - [Detailed Licensing](#) - [Undeclared](#)

This page titled [Making LibreTexts Interactive with Binder: A Practical Guide](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).