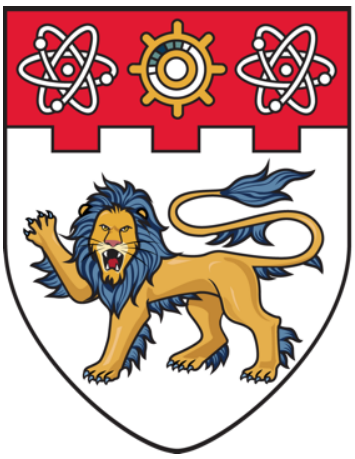


NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

PYTHON FRAMEWORK AND LIBRARIES TO DEVELOP DSP GUI DEMOS

NGUYEN THANH TUNG

Supervisor: A/P CHNG ENG SIONG

Examiner: A/P VINOD ACHUTAVARRIER PRASAD

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

2016/2017

NANYANG TECHNOLOGICAL UNIVERSITY

PYTHON FRAMEWORK AND LIBRARIES TO DEVELOP DSP GUI DEMOS

Submitted in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Computer Engineering
of the Nanyang Technological University

By

NGUYEN THANH TUNG

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

2016/2017

Abstract

The development of the Graphical User Interfaces (GUIs) has been found to be an essential component in learning of Digital Signal Processing (DSP) topics. In this project, we will be developing Python Framework and Libraries to develop DSP GUI demos. DSP GUI demos are software learning tools which helps students to understand DSP and/or Digital Communication concepts.

The aim of the Framework to integrate these Python GUI apps such that when the user runs the main code, the system will detect all the available Python GUI apps. The user can also add new Python GUI apps from other courses. Generally, this framework is a one-stop platform that can facilitate the various Python GUI apps from the different courses available. Also, to make this set of learning tool more versatile, we want the next user to be able to add new Python GUI apps without ever changing the main code. The structure of the one-stop platform was implemented by using an XML-based architecture.

Libraries are created to help user build the apps with ease. With the help of these libraries, the Python GUI apps are created short and more organized. These libraries contain basic widgets, helping tools such as plotting, animation, and layout customization.

This report will include the implementation of the Framework and describing the features of the Python GUI Libraries that have been developed. At the end of this project, several DSP Python GUI apps were developed to demonstrate the libraries and the framework. There will be examples for user on how to create Python GUI apps using the library and integrate into the framework.

Acknowledgments

This dissertation would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, my utmost gratitude to Prof.Chng Eng Siong, my final year project supervisor whose sincerity and encouragement I will never forget. Prof. Chng Eng Siong has been my inspiration as I hurdle all the obstacles in the completion this project.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Digital signal processing is an interesting topic. It is useful in many application fields of engineering and science. And if you get the fundamental ideas, it provides insight into many things we see in the world.

But you might not have a chance to learn about Digital Signal Processing unless you are in mechanical or electrical engineering fields. The problem is that most learning tools (books that classes use) present the concept of DSP starting with mathematical methods and abstractions. They are often theoretical, and few demonstrations of concepts are shown.

Besides, with the help of computers, programming and simulation tools can be used widely for helping students. The teaching methods nowadays have changed from lectures and books to lectures and laboratory environment, especially in engineering courses. Therefore, to make students understand concepts effectively, the computer-based demonstrations, examples must be used heavily.

There are currently many ways to do DSP demos. Besides, Python is a widely used high-level programming language for general-purpose programming; In term of scientific computing, Python and Python libraries are good enough to demonstrate concepts of DSP.

Python also supports developing applications with Graphical User Interface (GUI) features. GUI allows user to interact though graphical icons, and call various functions, parameters without having to touch the hiding codes. Nowadays, GUI are intensively used for basic users.

Using the GUI demos, the students can understand DSP better with explanations, examples. However, the Python GUI demos that have been developed were mostly limited, too complicated and separately (individual apps).

The framework developed in this project allows user to assemble all the apps available and navigate the apps without having to run them individually. With GUI, this Framework is a good learning tools for DSP course.

The libraries developed in this project allow user to create the apps with ease. The libraries contain many features that user does not need to know the actual native codes but to apply higher level of codes.

1.2 Objectives and Aims

Many individual DSP demos have been created, but most of them have limitations.

In the first contribution, we propose a framework for the multiple DSP Python GUI demos to be integrated such that it is easy for the user to navigate the apps. In the second contribution, we propose libraries to build up the apps and therefore, we can integrate these apps to the framework.

In this project, the following 2 objectives will be achieved:

- To develop a Python Framework: the key ideas are assembling all available apps, and navigating among them.
- To develop a Python libraries: Python GUI libraries help user to implement apps with less effort.

1.3 Report Organization

This report is divided into seven chapters and an overview of each chapter is as

follows:

- Chapter 1 provides an introduction of the project and the objectives of the project.
- Chapter 2 discuss on the related works that were used in this project
- Chapter 3 provides an overview of the proposed framework, libraries for DSP GUI demos and basic features of them
- Chapter 4 focus on the design of the framework for DSP GUI demos
- Chapter 5 discuss on the design of the libraries for DSP GUI demos and features that has been developed, some sample demos
- Chapter 6 concludes the report with future direction and lesson learnt.

Chapter 2

Literature Review

2.1 Related works on DSP

Overall, there are many demonstrations of DSP concepts. They are in form of books, figures, digital apps, web applets that many institutes use them to help students to understand DSP. Because of scientific and numerical characteristics of DSP, many scientific computing programming languages such as Fortran, MATLAB or even computational knowledge engine like Wolfram Alpha are used extensively to implement DSP applications.

MATLAB is the most popular proprietary programming language that is used to show examples of DSP or Digital Communication topics because codes are easily presented and better visualized.

However, in this project, we choose Python over MATLAB or any other languages to develop DSP demos. There are some advantages of Python:

- Python is a general-purpose programming language, it means we can use Python to do things other than scientific computing. Furthermore, Python is free and more popular than MATLAB
- Python is inherently object oriented, and fast enough to do heavy numerical computations.
- Python is high level, easy to learn, and fast to develop. Moreover, Python has great number of libraries that can be better utilized to developed applications.

In summary, Python is free, light, open source, no license server, full-featured modern object-oriented programming language and suitable for large-scale software development, Python packages can do nearly everything MATLAB can do for DSP.

2.2 Related works in Python on DSP GUI apps

Examples of Python demonstrations of DSP concepts are numerous.

One organized and well-known publication is “ThinkDSP (Digital Signal Processing in Python)” developed by Allen B. Downey, Massachusetts. The book discusses concepts such as Sounds and Signals, Harmonics, Noise, Autocorrelation, Discrete Fourier Transform, Filtering and

Convolution, LTI Systems. Each chapter of the book introduces a new technique and an application that can apply to real signals. At each step, the book shows with detailed explanations how to use a technique first, and then how it works. ThinkDSP covers almost every topics in DSP, with basic understandings. The limitation of this book is that the codes are in individual scripts, and they are not GUIs.

Another works in Python about DSP GUI apps is Pyo - Python DSP module by ajaxsoundstudio.com. The module containing classes for a wide variety of audio signal processing types. The limitation of the module is its focusing only in audio signal processing and the module is not GUI.

There are other scripts and modules about DSP GUI apps as well as more Python discussions and programs about DSP online. Like ThinkDSP and Pyo, these individual scripts are usually focus on one specific area of DSP such as audio processing, Fourier Transform and they are often not in structure of GUIs. Also, these available modules are usually in form of individual pieces of code, limited, not well-organized, difficult to navigate among them. Some of them are too complicated to build similar ones. So far, there is no simple framework to integrate Python GUI apps and no libraries to help writing Python GUI apps.

This project is to solve these limitations. First contribution, the Framework developed in this project will integrate available Python GUI apps. Also, the libraries can help the user to build app with less effort and great maintainability.

2.3 Project Similarity

The idea of DSP framework that developed in this project was influenced from GUIDE Educational Tool (GET) developed by Nonny Florentine (FYP SCE, 2016). The main difference is this framework is in Python, while Nonny's work is for MATLAB.

The DSP contents of the GUI apps were influenced by ThinkDSP mentioned above, and cover basic concepts such as Signals and Systems, Z Transforms.

Chapter 3

Proposed Python framework and libraries for GUI demos development

3.1 Proposed Python framework for GUI demos development

3.1.1 Framework Functionalities

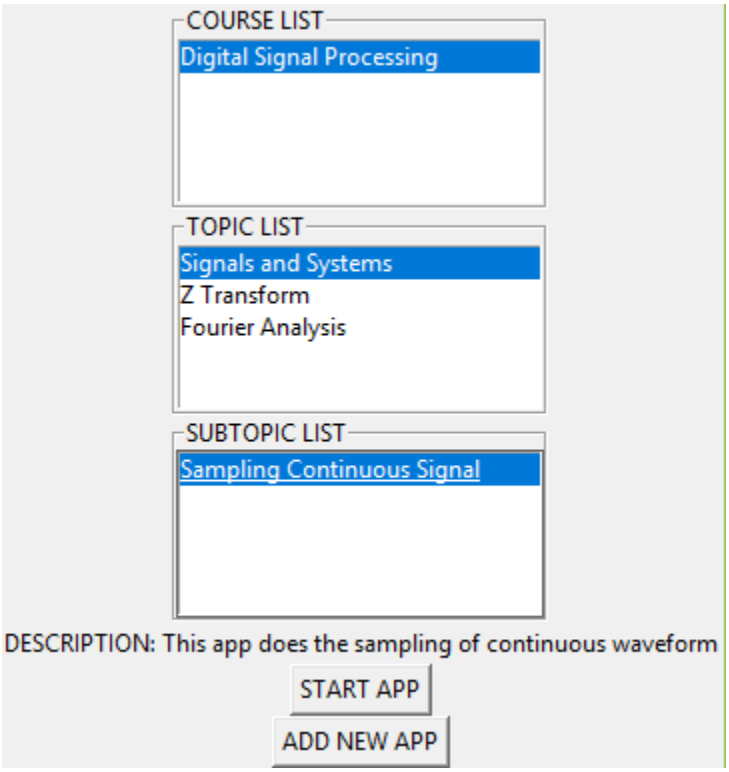


Figure 3.1: Framework GUI

Figure 3.1 shows GUI of the Framework. What we want to achieve in developing the framework is when the user runs the main code, the system will automatically search the available courses, topics and follow by the sub-topics to access the Python GUI demos. In addition, user did not need to modify any part of the code in the main code whenever new Python GUI demos are added. Thus we designed the demo's structure as shown in Figure 3.1 to achieve our objective.

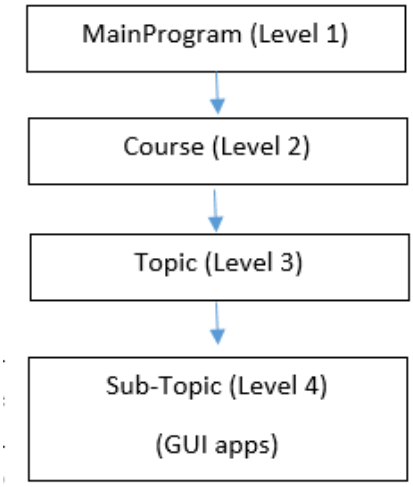


Figure 3.2 Structure of GUI demos

Each GUI demo has 3 attributes as 3 levels: Course – Topic – SubTopic to which it belongs. For example, in Figure 3.1, the demo "Sampling Continuous Signal" has 3 levels:

Course – Digital Signal Processing (Level 2)

Topic – Signals and Systems (Level 3)

SubTopic – Sampling Continuous Signal (Level 4)

Each GUI demo has different structures, as it is easier to categorized all of them, the structure of each GUI demo will be stored in XML files which will be discussed later.

The "COURSE LIST" list box contains all the courses available. Therefore, more courses such as Digital Communication can be added here.

When one of the courses are chosen (cursor clicked), The "TOPIC LIST" list box would display all the topics available under that course. In example of Figure 3.1, the course "Digital Signal Processing" contains topics: "Signals and Systems", "Z Transform" and "Fourier Analysis".

When one of the topics are chosen, the "SUBTOPIC LIST" list box would display all the subtopics or the GUI demos available under that topic. In example of Figure 3.1, the topic "Signals and Systems" contains topic "Sampling Continuous Signal".

The hierarchy of the GUI demos helps to easily categorize and navigate the GUI demos like expansion of a tree.

The summary of the Framework workflow can be illustrated in Figure 3.3 below

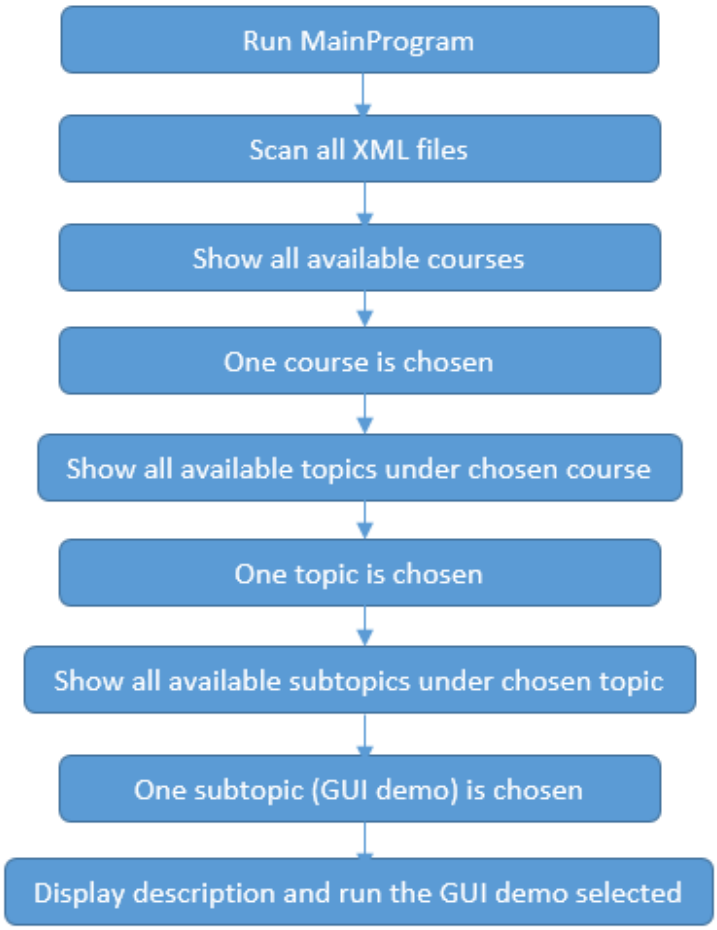


Figure 3.3 Framework Workflow

3.1.2 Adding new GUI demo app into the Framework

As mentioned before, the structures of the GUI demos are stored in XML files. The sample XML file is shown in Figure 3.4 below

```
<structure>
    <course>Digital Signal Processing</course>
    <topic>Signals and Systems</topic>
    <subtopic>Sampling Continuous Signal</subtopic>
    <appfile>Sampling</appfile>
    <directory>./Digital Signal Processing</directory>
    <mainclass>Sampling</mainclass>
    <description>This app does the sampling of continuous waveform</description>
</structure>
```

Figure 3.4 Sample XML template

The XML file contains all necessary tags for GUI demos. These tags details and explanations will be discussed in the design of the Framework.

After a create a new GUI demo, user can add the created GUI demo into the Framework by just creating a XML file supporting the main Python GUI demo file. To create this XML file, the user can manually create and edit the XML file following the structure of the template. Alternatively, the user can create using the Framework's "AddNewApp" module. In the Framework GUI (MainProgram), user can click on "ADD NEW APP" button to access this module. The GUI of this module is in Figure 3.5

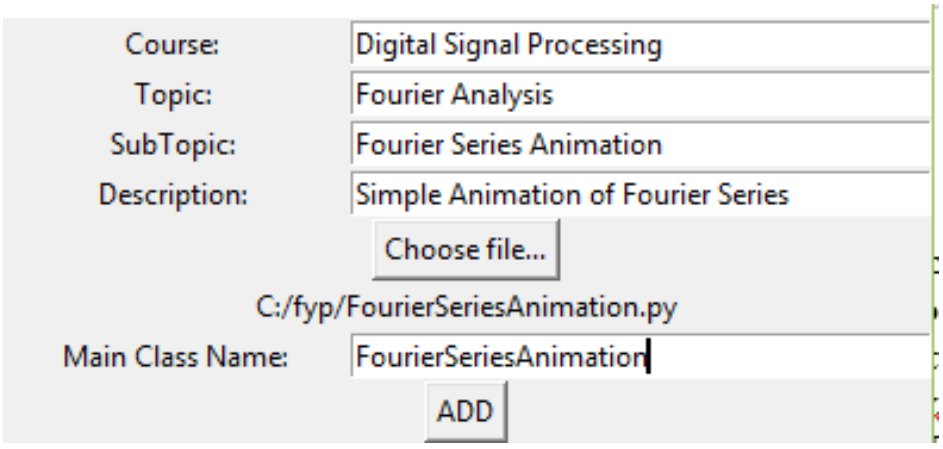


Figure 3.5 AddNewApp module GUI

Using this module, the user can add the created GUI demos into the Framework without touching the XML files. The module takes in the parameters in the entry fields and the browsed file to write an XML file with the same structure of template.

Note that when the GUI demo are added into the Framework, the course and topic to which the GUI demo belongs to will be added automatically in Course List and Topic List respectively. It is done by scanning of all XML files of the Framework. Hence there is no steps for adding course nor topics because they are added through adding GUI demos

3.2 The supporting libraries for Python DSP GUI demos development

The objectives of these libraries are to help user to create and build the individual GUI apps. Upon writing DSP GUI demos with native codes, it is found that native codes are repetitive, lengthy with many lines of code just to implement simple UI components. Alternatively, native codes can be used to implement simple GUI demos with few UI components. But with more complicated GUI demos, help of these libraries are recommended to make the codes shorter and more organized. The features and detailed design of these libraries will be discussed on Chapter

Libraries used codes

```
entry = ui.Entry(frame, defaultvalue='value', command=command,
                 width=20, height=50, side='top')
```

Native codes

```
entry = Tkinter.Entry(frame, width=20, height=50)
entry.bind('<Enter>', command)
entry.insert(0, 'value')
entry.pack(side='top')
```

Figure 3.6 Comparison between native and libraries used codes

Figure 3.6 is an example of comparison between native codes and libraries used codes.

In the example of Figure 3.6, if the GUI demos contains lots of entry fields, it would take more effort to write codes in native.

These libraries help user to interact with native only by parameters and method that are packed and simplified. With basic widgets in the libraries, user can build up the apps easily without knowing much about UI components in the native codes.

3.3 Summary of development of GUI demos using Framework and Libraries created

3.3.1 Definitions

In this project, the following definition of each key term was made:

DSP:	Digital Signal Processing, the core content of the apps developed
Python GUI app:	The learning tool (application) in the form of GUI (subtopic level)
MainProgram:	The main graphical user interface that act as a framework that integrates all the Python GUI apps
XML (*.xml) files:	Markup Language Files defined structure of the apps or simple layouts, templates of the individual apps
*.py files:	Main Python code of the individual apps
Numpy:	An extension to the Python programming language, used for basic numerical computation
Scipy:	Python library used for scientific computing and technical computing.
Matplotlib:	Plotting library for the Python and Numpy, used for visualisation.

3.3.2 Steps to develop GUI demos and integrate into Framework

- Develop individual GUI demos using supporting Libraries

- Add the created GUI demos into the Framework by “AddNewApp” module

Chapter 4

The design of the framework for GUI demos

The Framework is the learning tools mainly for DSP course. The main functionalities of the Framework are to integrate and list all available DSP GUI demos, categorize them in different topics and help user to navigate the DSP demos.

4.1 Development Strategy

Agile approach was used in this project, as every new Python GUI app that is being implemented has to be tested straight away after developing the program. Agile approach promotes adaptive planning and encourages rapid identification to problems at stage with its iterative working style. The agile methodology developed fast where project scope is flexible and requirements may change.

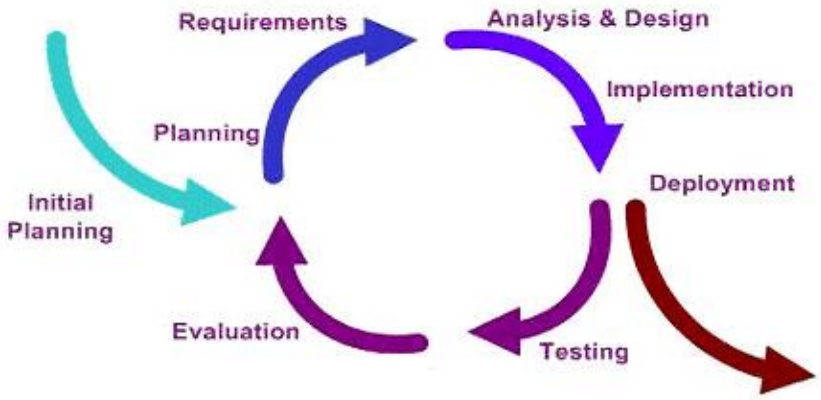


Figure 4.1 Agile Approach used in this project

4.2 Framework System Structure

The idea of framework structure is very simple.

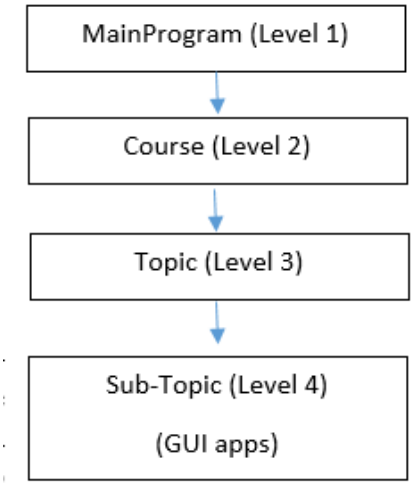


Figure 4.2 Framework Structures

There are 4 hierarchy levels in the structure – MainProgram, Course, Topic and SubTopic. The MainProgram consist of the main code (Main Program app in short MainProgram) and it functions as a platform to access the various Python GUI app demos. The Python GUI apps will be classified as level 3 that is the Sub-Topic. The Course and the Topic serves as a connection from the MainProgram to the Python GUI apps.

In this project, an XML-based system architecture is used. The framework will include the MainProgram as the platform to access all the Python GUI apps. The Python GUI apps are classified under the Sub-Topic level. The 1st level and 2nd level, which are the Course and the Topic respectively, serves as a connection to the Python GUI apps. To connect all the levels we will use *.xml file such that it will link the Python GUI apps according to which topic and course they belong to. We will discuss the implementation in detail about the *.xml file later.

4.3 The Framework Graphical User Interface:

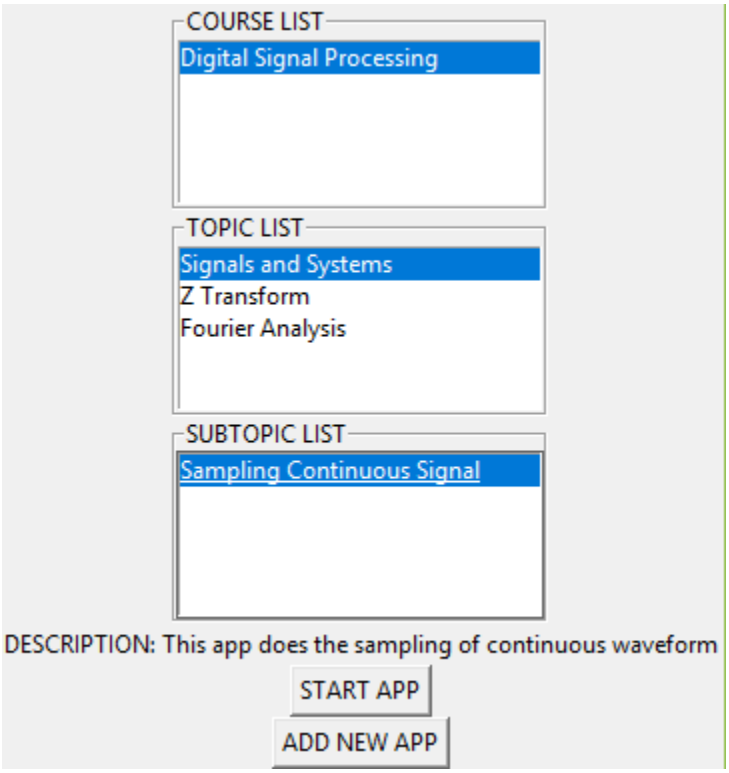


Figure 4.3 The Framework GUI

To implement the framework, firstly the MainProgram GUI is designed based on the supporting GUI libraries created that will be discussed later. Figure 4.3 shows the GUI of the MainProgram, it contains basic GUI components like listboxes, buttons, labels. The MainProgram provides a simple interface such that user can select the courses, topics and follow by the subtopics that include a description for the user to refer.

The GUI components in detail used are: 3 listboxes, static text, 2 buttons. The 3 listboxes are for the lists of courses, topics, sub-topics. The static text is for showing descriptions which each sub-topic has its own. One push button is for opening new Python GUI apps once the sub-topic (DSP GUI app) has been chosen. One push button is for adding new GUI apps into the framework (open AddNewApp Module).

4.4 Tools and Libraries

The table below provides an overview of the environment setup of this project:

Table 4.1 Environment setup

Operating System	Windows 10 Pro
Programming Language	Python version 2.7
Supporting packages and libraries	Anaconda 2 (for Python 2.7) Libraries: numpy, scipy, matplotlib, Tkinter
IDE for development	Spyder (in Anaconda packs)

4.4.1 Numpy + Scipy:

This project is developed using Anaconda from Continuum Analytics, which is a free Python distribution that includes all the packages needed to run the code (and lots more). Furthermore, by default, Anaconda does a user-level installation, not system-level, so it does not need administrative privileges. And it supports both Python 2 and Python 3. It can be downloaded from http://continuum.io/downloads.

If you don't want to use Anaconda, you will need the following packages:

• NumPy for basic numerical computation, http://www.numpy.org/;

• SciPy for scientific computation, http://www.scipy.org/;

• matplotlib for visualization, http://matplotlib.org/.

Although these are commonly used packages, they are not included with all Python installations, and they can be hard to install in some environments. If you have trouble installing them, Anaconda or one of the other Python distributions that include these packages is recommended.

4.4.2 Tkinter

The Tkinter module (“Tk interface”) is the standard Python interface to the Tk GUI toolkit. Tkinter is not the only GuiProgramming toolkit for Python. It is however the most commonly used one. It is also included by default with Python. Tkinter is known for its simplicity, easy to use and developed simple applications.

There are several alternatives to Tkinter:

- PyQt: Qt is probably the best cross-platform interface toolkit available. It’s stable, mature, cross-platform and completely native. Here are only two real (potential) disadvantages: PyQt is only available under the GPL or under a commercial license (which costs money); PyQt is also not included by default with Python installations, you're going to have to package the library yourself.
- wxPython: this GUI library provides more flexibility, you may want it if you need something more sophisticated. But it might be slower than Tkinter in the same applications.

In this project , Tkinter is used as the based library to build up the framework and framework library. So the individual apps must be developed based on Tkinter in order to integrate into the framework.

4.4.3 matplotlib

Matplotlib is a visualization library used for plotting figures in Python. It is most common used and powerful. You can typically do anything you need using matplotlib but it is not always so easy to figure out. It just takes too much work to get reasonable looking graphs

There are several alternatives to matplotlib, many of them are thin wrappers of matplotlib. Bokeh is a robust tool if you want to set up your own visualization server but may be overkill for the simple scenarios. Plot.ly generates the most interactive graphs; you can save them offline and create very rich web-based visualizations.

In this project matplotlib is used for basic visualization since very heavy tasks are not in the DSP apps.

4.4.4 Scope of the DSP GUI demos.

Since numpy and scipy are 2 standard libraries for numerical and scientific computation in Python, while matplotlib and Tkinter may have lots of alternatives. Therefore, in this project, we will focus on developing DSP GUI demos that are based on Tkinter and matplotlib. In other words, GUI demos based on other GUI and visualization libraries cannot be integrated into the Framework nor created by supporting GUI libraries.

4.5 The Framework Functionalities implementation

After specifying GUI components, we need to code the functionalties of the MainProgram. Before that, first we will again look at the workflow of the MainProgram.

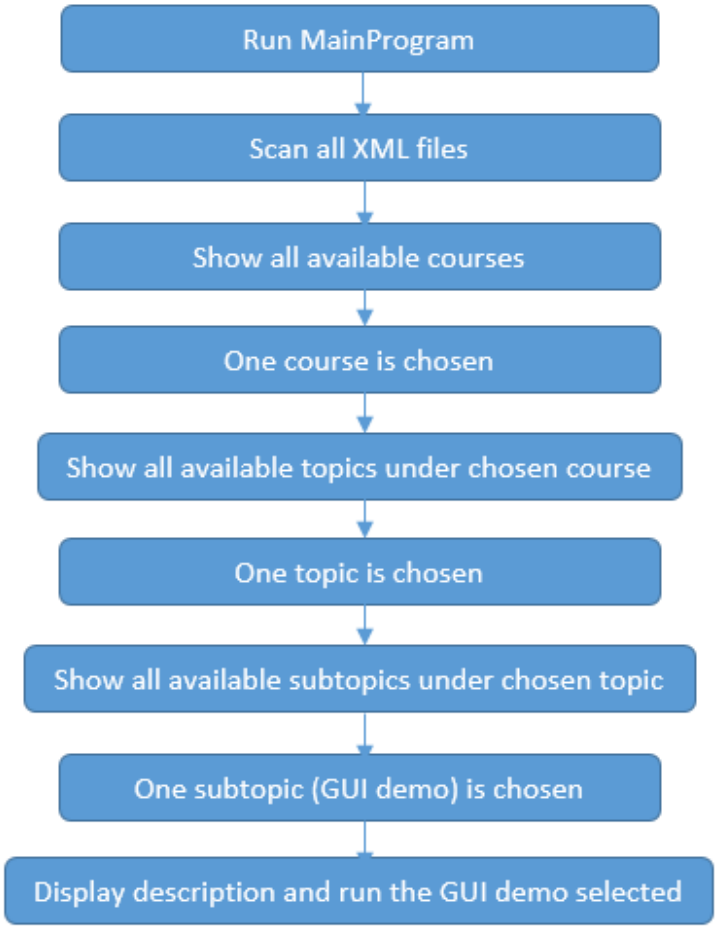


Figure 4.4 The MainProgram workflow

The Figure above shows the workflow of the MainProgram. When the user runs the MainProgram, the program (MainProgram.py) will automatically search for available courses, follow by the topics and sub-topics by scanning all the XML files in the directories and sub-directories. The list of topics will appear accordingly from the course chosen by the user. And list of sub-topics (GUI demos) will appear accordingly from the course and topic chosen by the user. Hence if a GUI demo is available, its following course and topic are also available.

We discussed earlier that an XML-based architecture is used to implement the Framework. We designed a simple template that serves our purpose in building the MainProgram. Each of the

levels (Course, Topic, Sub-topic) is assigned to one attribute in XML template. The template has 1 root with several supporting branches. The root tag is called <structure> and supporting branches are <course>, <topic>, <subtopic>, <appfile>, <directory>, <mainclass>, <description>. The sample template is shown in the Figure below:

```
<structure>
    <course>Enter course name here</course>
    <topic>Enter topic name here</topic>
    <subtopic>Enter sub-topic name here</subtopic>
    <appfile>Name of the .py Python file</appfile>
    <directory>Directory address of the Python file</directory>
    <mainclass>Name of class to start the demos</mainclass>
    <description>Description of the GUI demos</description>
</structure>
```

Figure 4.5 The template of the XML files

The following table explains the detail of each of the tags (branches)

Table 4.2 The input for XML tag

Tag	Input
<course>	Specify the course of the GUI demo followed (1st level)
<topic>	Specify the topic of the GUI demo followed (2nd level)
<subtopic>	Specify the name of the GUI demo (3rd level)
<appfile>	Name of the Python file of the GUI demo (excluding ".py")
<directory>	Directory address of the Python file (can be direct address or relatively address with MainProgram.py directory)
<mainclass>	The name of main class that runs the GUI demo
<description>	A brief description of the GUI demo developed

For each GUI demos developed, the user must strictly specify these above tags correctly. All the tags content must not leave blank except <description> tag. Note that there is no need for XML files for Courses and Topics levels, meaning adding one GUI demo XML files will automatically add its following course and topic into Course List and Topic List respectively.

To implement this functionality, 2 main functions are integrated in the MainProgram code. The 2 functions are findAllXMLFiles and processXMLFile. We use these 2 functions for the following purposes:

- findAllXMLFiles function is to scan all available *.xml files in the directory and sub-directories. And it only searches for XML files which has root tag <structure>.
- processXMLFile function is to parse and convert Extensible Markup Language (XML) file into a Python dictionary which is similar to an expanding tree of 3 levels (courses, topics, sub-topics) for easy accessing the data and building up the Framework structure.

The procedure to add new GUI demos are discussed in the previous chapter. Remind that, to add new GUI demo, the XML file can be written manually following the structure or AddNewApp can be used to create the XML file automatically.

5. The design of the libraries for GUI demos

These libraries are created to build up individual GUI demo with ease. These libraries hide the native Tkinter code so the GUI demo developers do not need to know much about Tkinter library but are still able to develop the GUI demo easily. Note that GUI apps can be developed using standard Tkinter library to integrate into the Framework. These libraries for GUI demos can be categorized into 2 types: layout generator library – for generating abstract frame layout for GUI demos, and GUI component library – for creating the UI components for GUI demos.

5.1 The implementation of libraries for GUI demos

These libraries are functions and classes based on Tkinter library, but heavily packed, these functions only interact with main apps through necessary and optional parameters. Usually, native Tkinter code would require more lines of code, and more unnecessary optional parameters.

5.2 Functionalities of the libraries for GUI demos

5.2.1 Layout Generating Library

The main purpose of this library is to translate XML of JSON contents into Tkinter frame layouts. A frame is rectangular region on the screen. The frame widget is mainly used as a geometry master for other UI components, or to provide padding between other UI components. Frames are used to group other components into complex layouts. They are also used for padding, and as a base class when implementing compound UI components.

For XML contents, the library takes as an element tree, and uses it to populate a Tkinter Frame widget. The function supports two kinds of elements: Frame elements and LabelFrame elements. Frame elements are used to create and populate Tkinter frame widgets. Each Frame element can contain one or more Frame elements. The Frame elements can have attributes width, height, color, paddings, etc…

Take an example of the XML contents:

```
<frame>
    <frame name='frame1' bg='blue' side='left'/>
    <frame name='frame2' bg='red' side='left'/>
    <frame name='frame3' bg='green' side='right'/>
    <frame name='frame4' bg='yellow' side='right'/>
</frame>
```

The example XML specifies that Blue and Red Frames are on the left side of the main window, Green and Yellow Frames are on the right side of the main window.

The translated layout can be shown in the below Figure

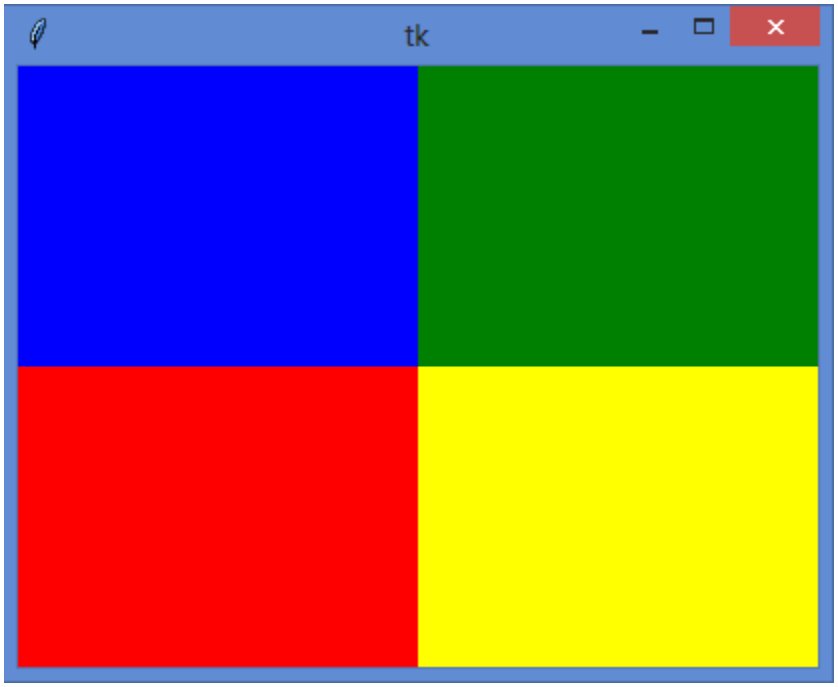


Figure 5.1 Example Layout

A simple or more complex layout can be generated this way to created abstract layout and later on, UI components can be added into each Frame.

For JSON definition files, the concept is similar, but just different presentation.

For the above layout in Figure 5.1, the following JSON file is used to generate the layout.

```
{
    "Frame":{
        "Frame":{
            "name": "Frame1",
            "side": "left",
            "bg": "blue"
        },
        "Frame":{
            "name": "Frame2",
            "side": "left",
            "bg": "red"
        },
        "Frame":{
            "name": "Frame3",
            "side": "right",
            "bg": "green"
        },
        "Frame":{
            "name": "Frame4",
            "side": "right",
            "bg": "yellow"
        }
    }
}
```

The attributes for Frame can be specified more: width, height, paddings (left, right, top, bottom), background color, border. These attributes details are in the documentation.

5.2.2 UI components library

The library provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets or UI components. The library includes general purpose widgets such as buttons, textboxes and DSP-related widgets such as waveform, LTI systems.

The details of these general-purpose UI components are in following table. Further details of initialization, parameters, methods are in documentation.

Table 5.1 – Description of general-purpose UI Components

UI components	Description
Label	The Label widget is used to provide a single-line caption for other widgets. It can also contain images.
Entry	The Entry widget is used to display a single-line text field for accepting values from a user.
LabelEntry	The LabelEntry widget is used to display a single-line caption along with a single-line text field for accepting values, i.e. combining of Label and Entry widget.
Button	The Button widget is used to display buttons in your application.
Checkbutton	The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
Listbox	The Listbox widget is used to provide a list of options to a user.
Radiobutton	The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
Figure	The Figure widget is used to display a plotting of graph, image, and other visualizations. This Figure widget is based on matplotlib mentioned before. The Figure widget support multiple plottings and animation as well.
EntryScale	The EntryScale widget is used to provide an entry to display a text field along with a slider widget. The number in entry and slider value are updated whenever either one is changed.

These general-purpose GUI components have same functionalities as the standard Tkinter UI components. The difference between this library and standard Tkinter library is that this library pack with necessary parameters and methods in few lines of code, while Tkinter uses longer and more complicated code to describe same UI components. This library hides all these long and complicated Tkinter functions, and interact with main app through necessary parameters and methods, functions. The comparison for these libraries can be shown below:

For creating a simple label or a display box where you can place text

Figure 5-2 Simple Label Component

These following codes are used to implement the above result:

Standard Tkinter code

`var = StringVar()`
`label = Tkinter.Label(Frame, textvariable=var)`
`var.set("Hey!? How are you doing?")`
`label.pack()`

Using library for GUI demos
`label = UIComponents.Label(Frame, text='Hey!? How are you doing?')`

Take another example of radio buttons

Figure 5-3 Simple Radiobutton component

To create a simple radiobutton option 1-3, whenever an option is selected, the label would change, the following code is used

Standard Tkinter code

def sel():
  selection = "You selected the option " + str(var.get())
  label.config(text = selection)
var = IntVar()
R1 = Tkinter.Radiobutton(Frame, text="Option 1", variable=var, value=1, command=sel)
R2 = Tkinter.Radiobutton(Frame, text="Option 2", variable=var, value=2, command=sel)
R3 = Tkinter.Radiobutton(Frame, text="Option 3", variable=var, value=3, command=sel)
label = Tkinter.Label(Frame)
label.pack()

Using library for GUI demos

def sel():
  selection = "You selected the option " + str(radiobutton.get())
  label.change_text(selection)
R = UIComponents.RadioButton(Frame, [‘Option 1’, ‘Option 2’, ‘Option 3’], command=sel)
label = UIComponents.Label(Frame)

For the GUI demos developers, they does not need to know much about standard Tkinter syntax. Using this library for GUI demos, the developers would take less time and effort to implement same GUI components. The code looks neat, short, and easier to organize these components in the app. Beside initializing components, other methods would be optimized as well such as plotting, animation for figure, updating values for text and figures. The details of these classes and functions are further in documentation.

Besides these general-purpose UI components, other components related to DSP course are created to build the DSP GUI demos. The detail descriptions are in the table below:

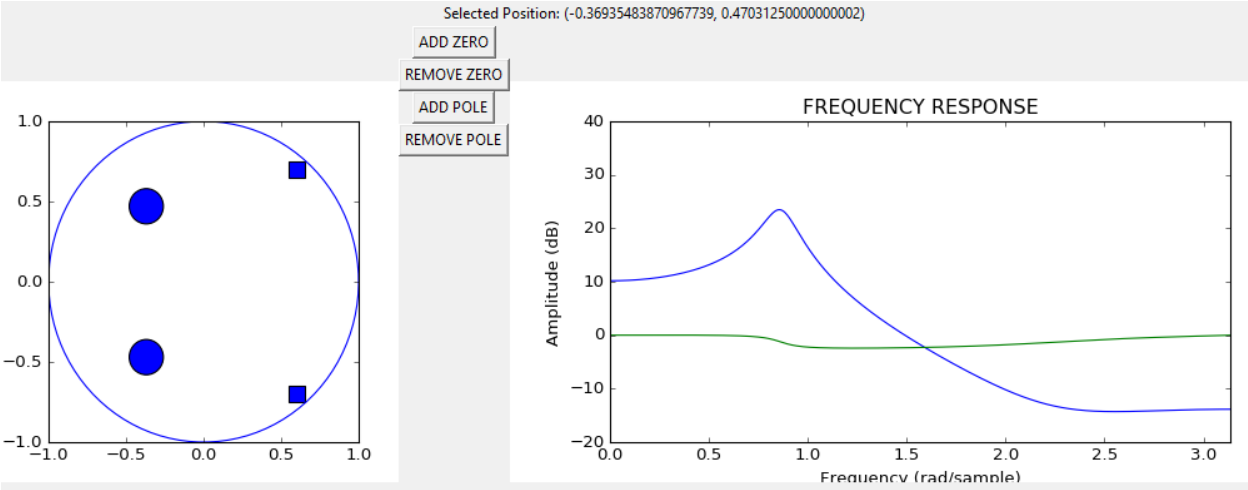
Table 5.2 Description for DSP widgets

DiscreteSignal	The DiscreteSignal widget is used to display a simple selection for discrete signal. It includes Sine, Exponential, Pulse, Unit Sample, and User Signal choices. For each type of signal, different parameters are used to describe the signal. A figure is shown to display the signal chosen.
System (LTI)	The System widget is used to display a simple selection for LTI system. It includes Zero-Pole-Gain, and Coefficient types. For each type of LTI system, different parameters are used to describe the system.
Waveform	The Waveform widget is used to display a simple selection for Waveform Signal. It includes Sine, Square, Sawtooth, Triangle and User Signal (experimental) for waveform. For each type of waveform, different parameters are used to describe the waveform. A figure is shown to visualize the waveform
DraggablePoint	The Draggable widget is used to display a point (shape of circle, rectangular) on the Figure widget. When the Point is being dragged, other components such as Labels, Figures can be updated concurrently.
Animation	This function is included in the Figure widget. It creates a simple animation on the figure. Parameters are needed to specified such as plotting functions, timestamps

5.3 Example Python GUI demos

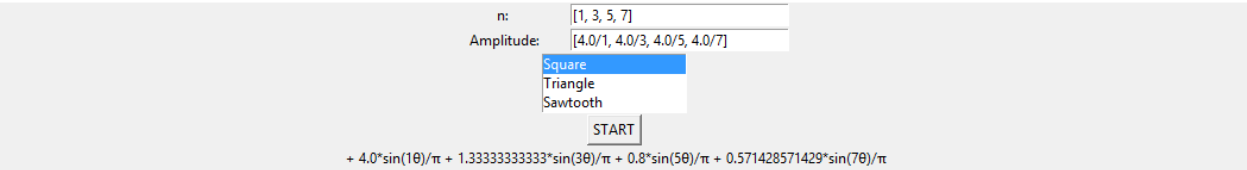
The three topics under DSP course that has been developed are Signals and Systems, Z-Transform and Fourier Analysis.

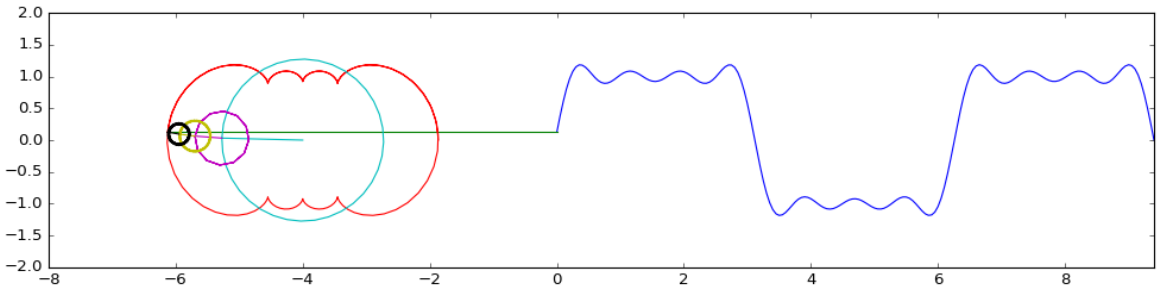
a. Z-Transform zero and pole dragging graphics



The Z-Transform zero and pole dragging graphics app is an application where user can drag the zeros and poles in the graph and the magnitude graph will update automatically.

b. Fourier series animation



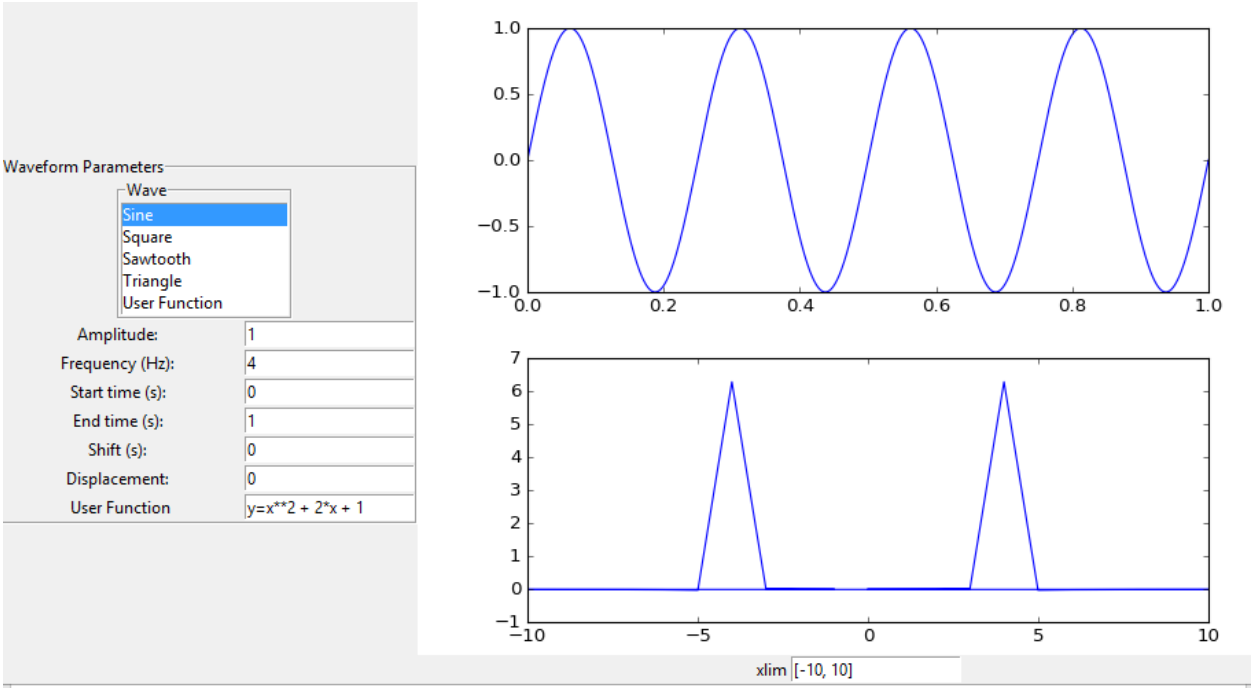


The Fourier series animation is an application demonstrating Fourier series representation of periodic signals as a sum of sinusoidal functions. The features of the app:

• Users can change the period, amplitude and frequency values.

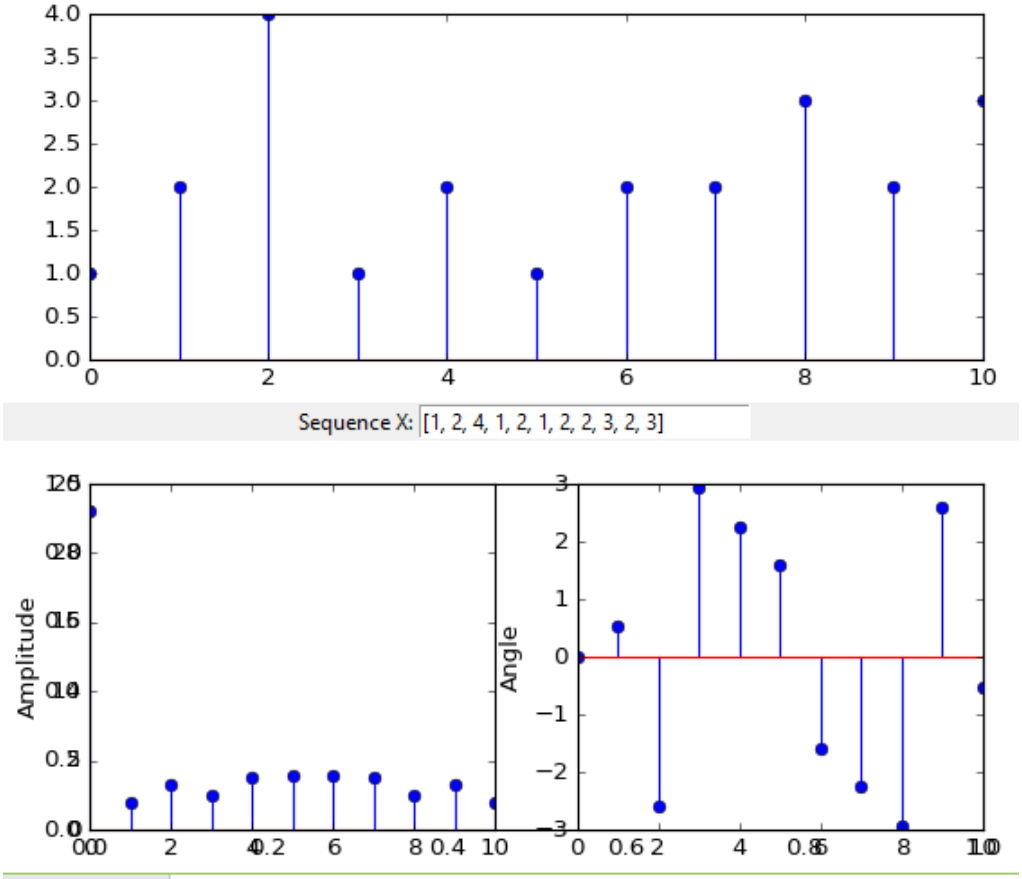
• Simple animation that only plays when the play button is pressed.

c. Continuous Time Fourier Transform



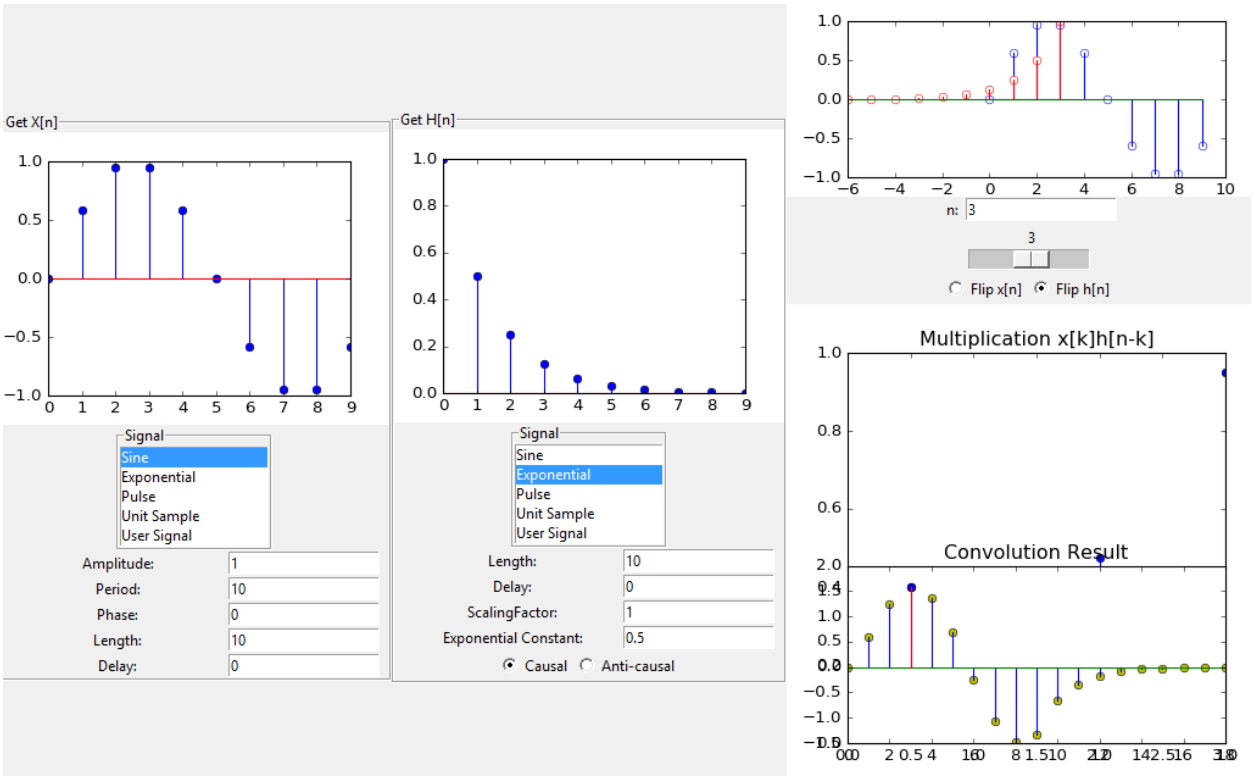
Using the widget Waveform mentioned earlier, the app uses this waveform with various choices and parameters to create Continuous Time Fourier Transform.

d. Discrete Fourier Transform



The Discrete Fourier Transform app is an application that allow users to visualize the magnitude and angle of the Fourier Coefficients.

e. Discrete-time Convolution



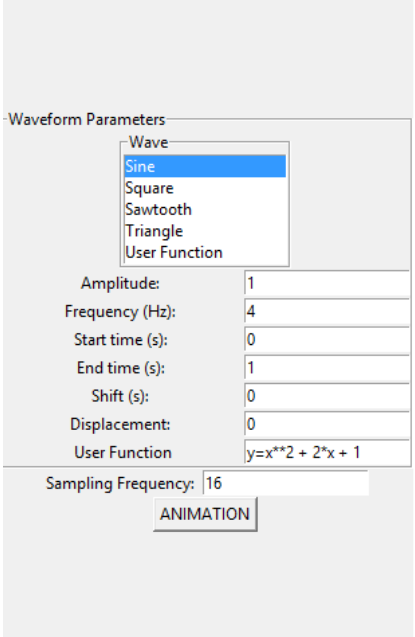
The Discrete-time Convolution app is an application that helps visualize the process of discrete-time convolution. The features:

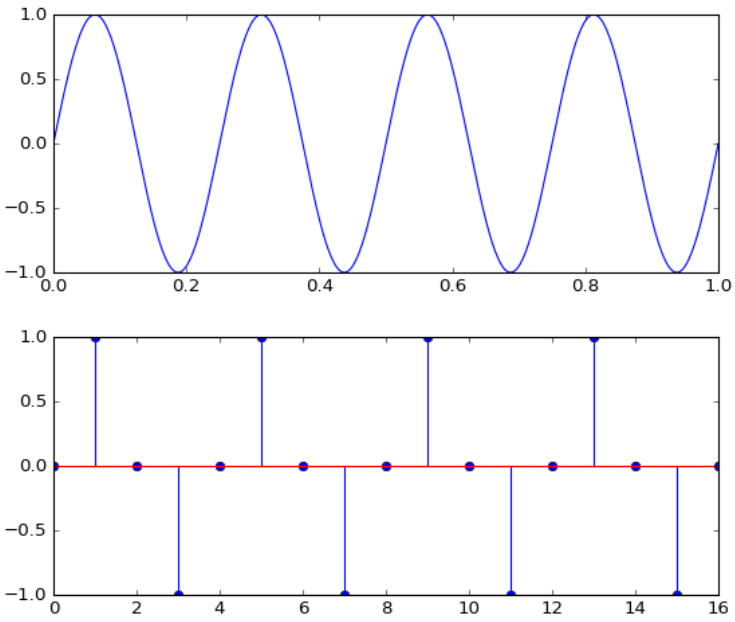
• Users can choose from a variety of different signals: sine, cosine, exponential, pulse, unit sample and user signal.

• Users can observe in the multiplication and the linear convolution graph as the n input value is being shifted

• Various plot options enable the tool to be effectively used as a lecture aid in a classroom environment.

f. Continuous-time Signal x(t) to Discrete-time Signal x[n]





The Continuous-time Signal x(t) to Discrete-time Signal x[n] app is an application that allows to create a set of periodic continuous time signals and convert to discrete time signals. The features:

• Users can change the input frequency and sampling rate values.

• Able to show the value of x[n] on the graph.

6. Conclusion

6.1 Summary of Achievements

In this project, the Framework for DSP GUI demos was implemented. When the user want to add new Python GUI apps, no changing of the main code is needed. The user simply follows the procedure on how to add new Python GUI app.

The libraries are also implemented to aid the development of GUI apps. With layout generator, common GUI components and several DSP related widgets, developers can use these libraries to create useful DSP demos.

In conclusion, several DPS Python GUI apps were developed. The MainProgram is able to do an automatically search for all the Python GUI apps that are available in the Framework system.

6.2. Reflection and Future Work

In this project, future research should explore following directions:

1) Improvement on the design and the functionality of the Framework;

2) Improvement on the Python DSP GUI demos

3) Development of a new Python GUI apps for the Digital Communication course.

4) Improvement on the supporting libraries for the Python DSP GUI demos, adding new features (audio and image processing, more UI components and DSP components)

References

[1] James H. McClellan, Mark A. Yoder, and Ronald W. Schafer. ***Signal Processing First,*** 1st edition. Pearson/Prentice Hall, 2003

[2] Using MATLAB to develop standalone graphical user interface (GUI) software packages for educational purposes, retrieved from: http://cdn.intechopen.com/pdfs-wm/11608.pdf

[3] Alan V. Oppenheim, Ronald W. Schafer, John R. Buck, Discrete-Time Signal Processing, Prentice Hall, ISBN 0-13-754920-2

[4] XML Tutorial, retrieved from: http://www.w3schools.com/xml/xml_tree.asp

[5] John G.Proakis and Dimitris G. Manolakis. ***Digital Signal Processing: Principles, Algorithms and Applications,*** 3rd edition. New Jersey: Prentice Hall, 1995

[6] Sanjit K. Mitra, Digital Signal Processing: a Computer-Based Approach, 2nd edition. Irwin: McGraw-Hill, 2001

[7] Think DSP: Digital Signal Processing in Python Book by Allen B. Downey

[8] Adventures in Signal Processing with Python, retrived from https://www.embeddedrelated.com/showarticle/197.php

Appendix A

```
def findAllCourse(self):
        courseList = []
        for item, parent in MainProgram.Structure.items():
            if parent == 'course':
                courseList.append(item)
        return courseList

def findAllXMLFiles(self):
        matches = []
        for root, dirnames, filenames in os.walk('.'):
            for filename in fnmatch.filter(filenames, '*.xml'):
                matches.append(os.path.join(root, filename))
        return matches

class AddNewApp:
    def __init__(self, master):
        self.master = master
        self.courseEntry = ui.LabelEntry(master, text='Course: ',
      labelwidth=20, entrywidth=40)
        self.topicEntry = ui.LabelEntry(master, text='Topic: ' ,
      labelwidth=20, entrywidth=40)
        self.subTopicEntry = ui.LabelEntry(master, text='SubTopic: ' ,
      labelwidth=20, entrywidth=40)
        self.descriptionEntry = ui.LabelEntry(master, text='Description: ',
      labelwidth=20, entrywidth=40)
        self.browseButton = ui.Button(master, text='Choose file...',
      command=self.openfile)
        self.fileLabel = ui.Label(master)
        self.mainClassEntry = ui.LabelEntry(master, text='Main Class Name:' ,
      labelwidth=20, entrywidth=40)
        self.addAppButton = ui.Button(master, text='ADD',
      command=self.addApp)
```

```
def openfile(self):
    self.file = tkFileDialog.askopenfile(parent= self.master
  ,mode='rb',title='Choose a Python file')
    self.fileLabel.change_text(self.file.name)

def addApp(self):
    try:
        appfile = os.path.split(self.file.name)[1][0:-3]
        directory = str(os.path.split(self.file.name)[0]).split('/')
        cwd = os.getcwd().split('\\')
        relativeDirectory = directory[len(cwd):]
        string = '.\\' + '\\'.join(relativeDirectory)
        root = ET.Element("structure")
        ET.SubElement(root, "course").text = self.courseEntry.get()
        ET.SubElement(root, 'topic').text = self.topicEntry.get()
        ET.SubElement(root, 'subtopic').text = self.subTopicEntry.get()
        ET.SubElement(root, "appfile").text = appfile
        ET.SubElement(root, "directory").text =string
        ET.SubElement(root, "description").text
  =self.descriptionEntry.get()
        tree = ET.ElementTree(root)
        tree.write(string + "\\ "+ appfile+".xml")
        tkMessageBox.showinfo("Add New App", "Adding Successfully")
    except: pass
```