Developing iDSP apps in Python, with the STREAMLIT library

StreamLit is a python library which defines a large set of GUI (graphical user interfaces) to create reactive applications. The most useful are switches, sliders, test inputs. It also redirects the outputs of many python libraries (such as matplotlib) to a web browser.

Added to that, Streamlit-based applications, when installed on GitHub, can very easily be transformed into a publicly available web page.

These two features make StreamLit an ideal candidate for developing iDSP apps. In the next paragraphs, we give a simple example of how to create a simple but effective iDSP app: sinus.py.

1.1 Installing streamlit

Assuming the Anaconda distribution of python is installed on the computer, StreamLit is installed easily [1]by running the Anaconda Prompt app, and typing:

```
>>conda create mystreamlit
```

This will create a new (bare) python environment, which must mow by populated by useful libs:

```
>>pip install streamlit
>>pip install myplotlib
>>pip install scipy
>>pip install soundfile
```

1.2 Creating sinus.py

In a text editor, simply type:

```
import streamlit as st          # This must always be the first line of a streamlit app
from numpy import *
from matplotlib.pyplot import * # Needed for plotting

st.title('My first sine wave')
a=st.slider('Amplitude a', 0.0, 10.0, 5.0)
f=st.slider('Frequency f [Hz]', 0, 1000, 440)
phi=st.slider('Phase phi [rad]', -pi, pi, 0.0)

fe = 10000;                     # These are the three "computational" lines
t = arange(0.0,1,1/fe)
signal = a*sin(2*pi*f*t+phi)

st.latex('''a \sin(2 \pi f t + phi)''')
fig,ax = subplots(figsize=(10,4))
xlim(0,0.010); ylim(-10, 10)
```

[1] A more complex (but more complete) installation procedure can be found at https://docs.streamlit.io/library/get-started/installation.

```
plot(t[0:100], signal[0:100])
xlabel('Time (seconds)')
st.pyplot(fig)
st.audio(signal,sample_rate=fe)
```

There is not much to say about these lines, which are quite self-explanatory.

Notice that no callback function is needed (as is usually the case when creating real-time apps). This is mad possible in StreamLit by the fact that, whenever an input GUI is modified by the user, the whole app is run again (keeping all the values from GUIs of course)[2].

This file must then be run inside Anaconda, by typing:

```
>>conda activate mystreamlit
```

Or, for Windows, from the *Anaconda Navigator*, click on *Environments*, then click on the arrow on the right of mystreamlit and choose *OpenTerminal*. Then type:

```
>>cd .\path to the sinus.py file
>>streamlit run sinus.py
```

This will produce the screen shown on Fig. 1, running your app on your local browser, communicating with your local host.

1.3 Deploying the app

If we want the app to be used by others, we must now ask StreamLit to create a web page out of it. We must first create a Github repository for sinus.py.

We have done it here, as an example : https://github.com/thierrydutoit/sine-wave

The repository should include 3 files:

- sinus.py
- requirements.txt, a test file containing the list of libs required by the app:
  ```
  numpy
  matplotlib
  soundfile
  streamlit
  ```
- packages.txt, a text file required for playing sounds, containing simply:
  ```
  libsndfile-dev
  ```

When this is done, going to the localhost app and opening the *Deploy* menu on the top right will show a “StreamLit Community Cloud” option which will let you define the name of your github repository, and deploy the app automagically.

Assuming you decide to call it “sine-wave”, your app will then be available as:

https://sine-wave.streamlit.app/

[2] Since this may lead to excessive and useless computation load, StreamLit has a very efficient way of avoiding parts of the app to be re-run.

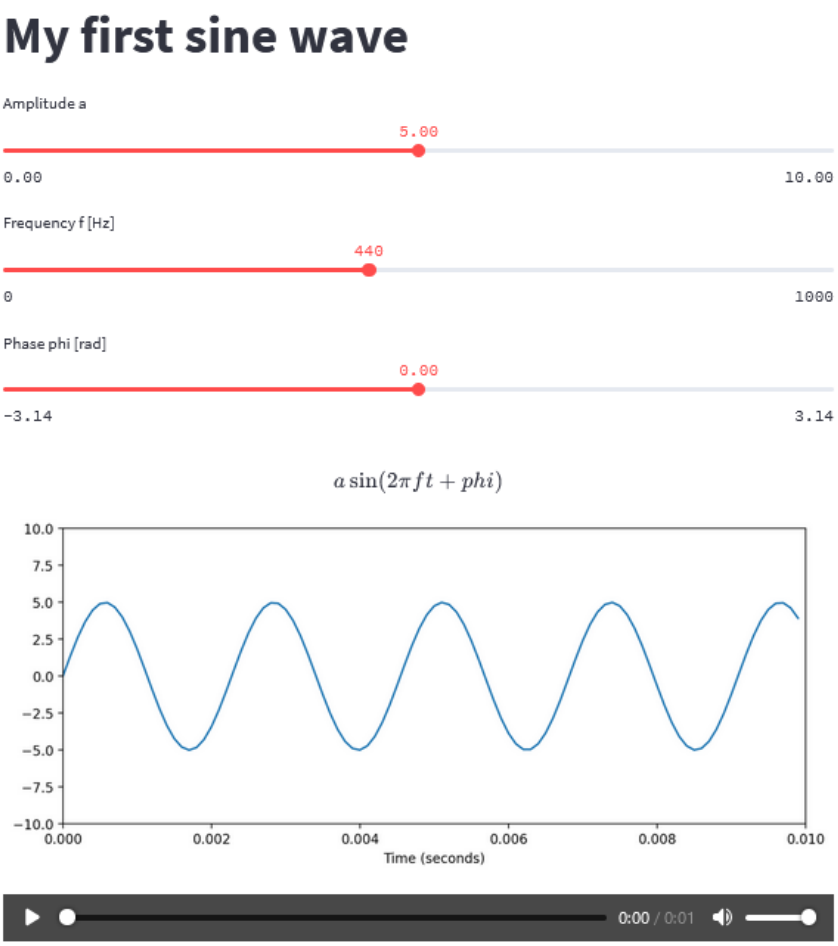

Fig 1. Sinus.py running on the local host web browser