Lab 1: Intro to GNU Radio

1 Objective

The objective of this lab is to get you familiar with GNU Radio Companion, including a number of basic blocks you will commonly use in future labs. Additionally, you will explore various linear and non-linear signal operations and observe how the different operations affect the signal.

1.1 Deliverable

There are six questions in Lab 1. At the top of your answer document, if you are working with a partner, make sure you include **both names** in your group. Turn in your answers to Blackboard in PDF format. You must also submit your **final** flow graph (.grc file) - corresponding to Questions 4-6.

2 Block Navigation

On the far right of the interface is a tree featuring the possible blocks. There is an eyeglass icon on the upper bar that will filter those block options by name. This should speed up finding particular blocks.

Figure 1: Highlighted Search Icon

3 Block Descriptions

Below are the descriptions of some of the most commonly used blocks in GNU Radio Companion. Read through them to gain a better understanding of each block before you start the lab and use them as reference points to come back to as needed.

3.1 Signal Source

The `Signal Source` block generates a signal in the shape of a particular wave. The different wave options are Sine, Cosine, Saw, Square, and Constant. Additionally, you can

specify the Amplitude, Frequency, Offset, and Sample Rate of the signal. The *Frequency* field is measured in Hertz (Hz).

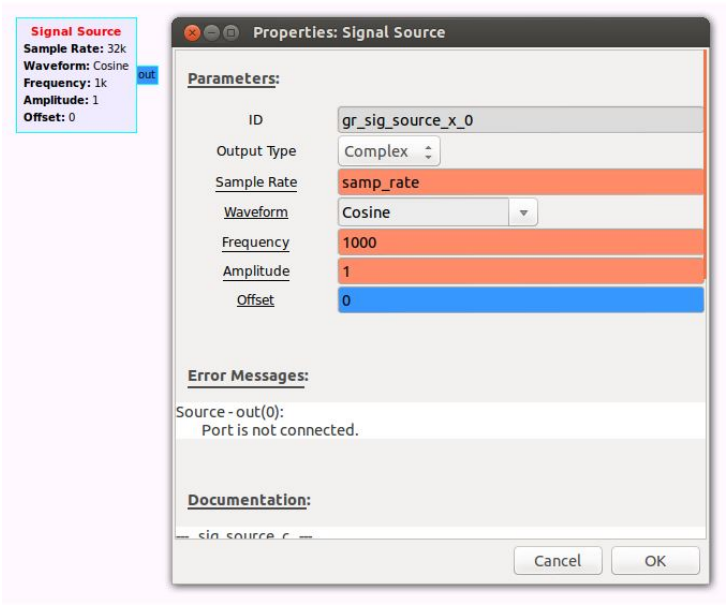

Figure 2: Signal Source Block

3.2 Wav File Sink

The `Wav File Sink` block is a block that will interpret its input signal as an audio wave form. It will write actively to a specified .wav file. While the flowgraph runs, it will continue to add data to the file, **so remember to kill the flowgraph after a few seconds**. There are two parameters: the *Sample Rate* and *File*. This specifies the rate at which the audio stream is flowing in and the name of the file it will write to. If the setting of the *Sample Rate* parameter does not match the actual sample rate of the incoming signal, the audio will become distorted. Clicking the three dots next to the file name on the file sink block will open a menu showing you where the file is saved.

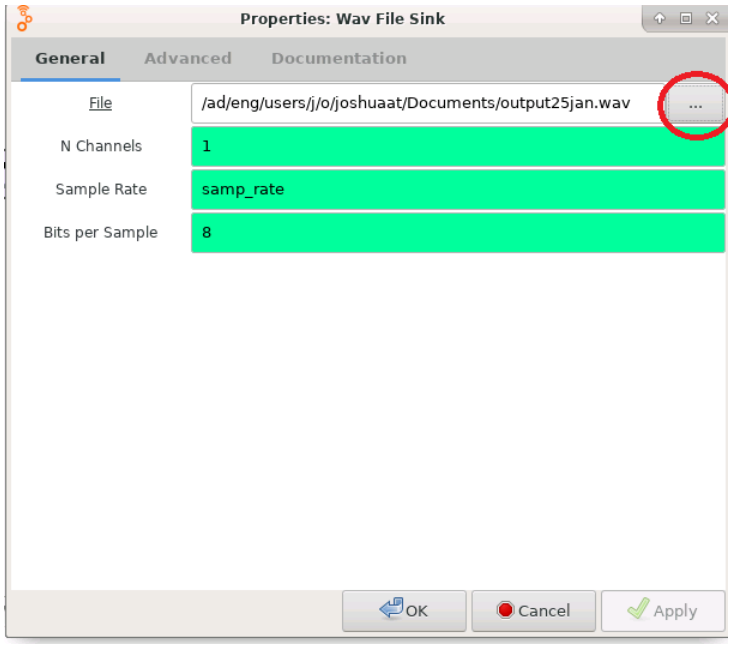

Figure 3: Wav File Sink Block

3.3 Math Operators

The Math Operator blocks are blocks used to perform operations on one or more signals to combine, modify, or select signals to create a different output. There are two types of Math Operators that we will use in this lab:

3.3.1 Signal-Signal Operations

These blocks take in one or more signals and will perform the operation on each signal to combine them into one output. Examples of these include the `Multiply` and `Add` blocks.

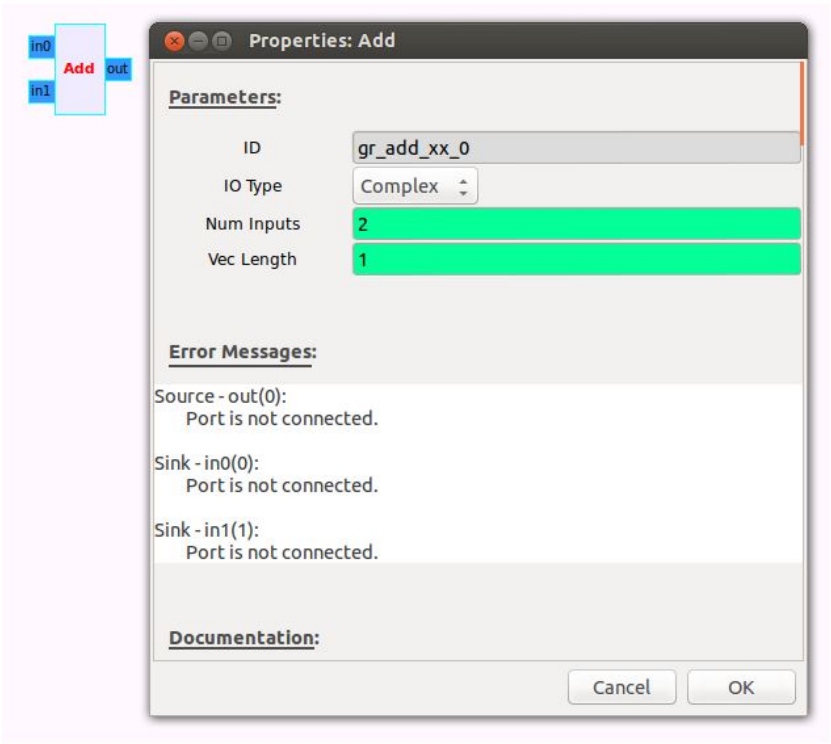

Figure 4: Add Block

3.3.2 Signal-Const Operation

These blocks take in one signal and will perform one of the basic math operations. However, unlike the Signal-Signal Operations, they instead modify the signal by a constant. Examples of these include the `Multiply Const` and the `Add Const` blocks.

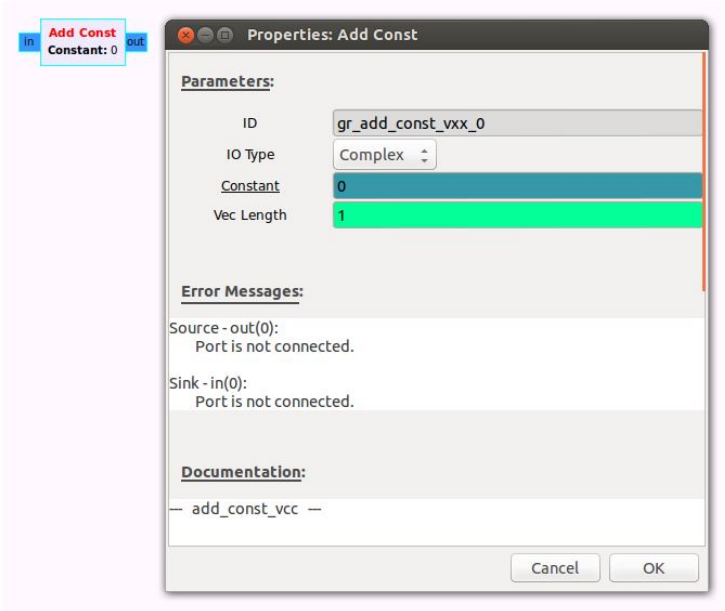

Figure 5: Add Const Block

3.4 Variables

The `Variable` blocks are blocks that will map a value to a given name. Changing the value of the block will pass that updated value to all blocks utilizing that variable. This provides a simple way to modify parameters that are used in several blocks, without having to manually modify each block.

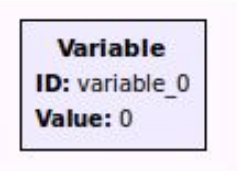

Figure 6: Variable Block

4 Procedure

Open the GNU Radio Companion by typing the following in the terminal:

```
gnuradio-companion &.
```

4.1 Creating your first flow graph

The simplest flow graph consists of a source and a sink. In this case, we will be using signal sources to generate a sine wave and a wav file sink to output the wave as sound.

4.1.1 Options Block

In these labs, we will be using the QT GUI library to graph our signals and waves. This setting controls the way that GUIs are generated for flow graph output. We must specify which GUI library to use in the `Options` block.

1. Right click the `Options` block in the top left corner and select 'Properties.' Alternatively, double click the block.
2. Change the *Generate Options* field to 'QT GUI.'
3. Click 'Apply' and then 'OK' to save your options.

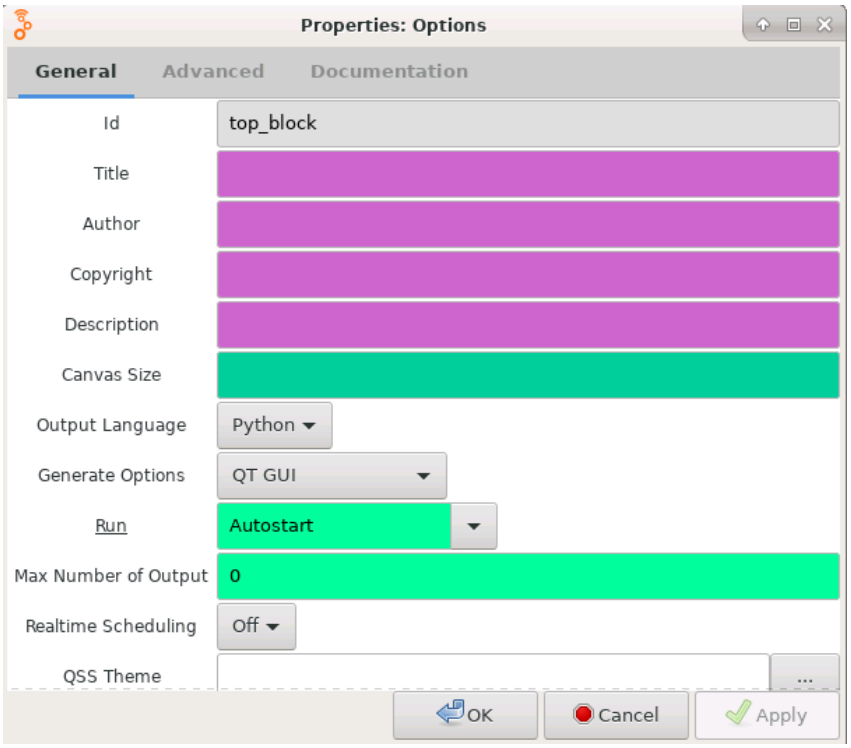

Figure 7: Options Block

4.1.2 Signal Source

1. Add a `Signal Source` block to the flow graph.
2. Once you find the `Signal Source` block, add it to your workspace by either dragging and dropping it or by double-clicking on it.
3. Open up the block properties by double-clicking on it and change the *Frequency* to 196 Hz. 196 Hz happens to be the musical note G.

4.1.3 Wav File Sink

1. Add a `Wav File Sink` to your flow graph.
2. At this point, you will have a Signal Source and a Wav File Sink However, one has a blue port while the other has an orange port. In GNU Radio Companion, data types are distinguished with colors on the ports as seen in Figure 8.

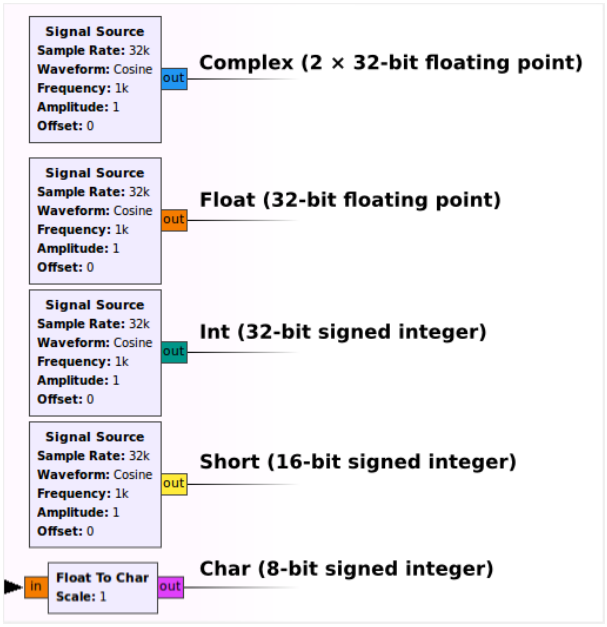

Figure 8: GNU Radio Block Types

You can access the full list of colors by finding the **Help** button on the top of the GNU Radio Toolbar and clicking **Types** as seen in Figure 9.

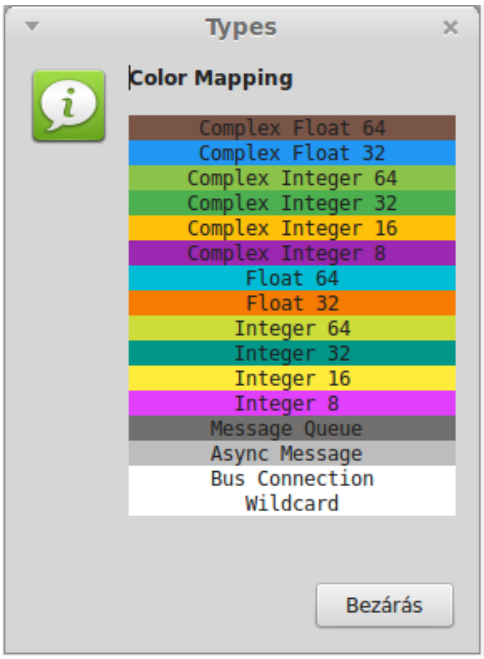

Figure 9: GNU Radio Block Types by Color

You cannot connect two ports with different types, they must be the same! (Fig 10)

3. There are two different ways to change the block type:

    1. Double click on the block and find the *Output Type* field and click on it. For the `Signal Source` block, you will find four different output types. The

`Audio Sink` receives **Float** as an input (orange). Therefore, set the output of `Signal Source` to be **Float**.

2. Click on the block you wish to change **once**. This should not pull up the properties. Press the up or down buttons on your keyboard until they match the color you wish for your blocks to connect.

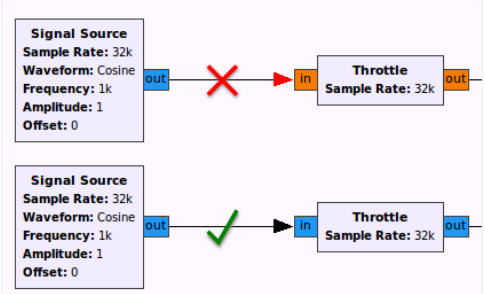

Figure 10: Correct Connection Types

4. Connect the `Signal Source` block to the `Wav File Sink` by clicking on the orange tab of one of the blocks and clicking on the other. An arrow should appear signifying that the blocks are connected.

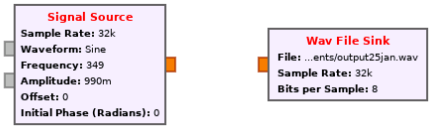

Figure 11: Connecting Blocks in GNU Radio

5. Save your flow graph by clicking on File/Save or by pressing the Save icon. Name your file *ec415_lab1_username1_username2.grc*.

6. Click the *Execute the flow graph* button on the top of the window. It looks like a play button.

7. **After a few seconds, click on the stop button. It is the square to the right of the play button.** At this point, the flowgraph has written this output to a file. The .wav filetype is often used for storing audio files, which we have just generated. Since we have remotely connected to the terminal we are unable to hear the sound through our local speakers.

8. Navigate to wherever you had the block save your .wav file. Use a service like Google Drive, Dropbox, or GitHub to upload this file so you can access it on your local machine (we recommend setting up a folder on one of these services for your labs, since you and your partner will be collaborating and sharing files/code)

   (a) If you are familiar with other file transfer software such as MobaXterm or Filezilla, you can use these as well. A short snippet on how to transfer files using scp is below.

(b) *How to use scp*: On windows, mac, and linux you can use the scp (secure copy) command to copy files to and from eng-grid and local machine. In a terminal (Windows Powershell/CMD, Mac/Linux Terminal), run the following command to transfer the file back to your local computer:

```
scp username@eng-grid2.bu.edu:PATH_TO_FILE SAVE_PATH
```

For example, if the lab file is on your desktop on the VLSI machine and is named *lab1.wav*, you could transfer it back to your local machine by using the following command on your local machine's terminal. The save path is set to a single period to represent the current working directory:

```
scp username@eng-grid2.bu.edu:Desktop/lab1.wav .
```

9. Many Operating Systems can natively play .wav files as audio, but we can also use a programming language to interpret them.

   In Matlab, they can be played with: .

```
filename = 'output.wav'
[y, fs] = audioread(filename);
soundsc(y, fs);
```

   In Python:

```
from playsound import playsound
playsound('output.wav')
```

10. Note that in these cases the file will need to be in the same directory as your script. It is possible but more complicated to include the path to the file if for some reason you need to store them separately.

11. If you hear a clicking noise, try changing the *Amplitude* field of the `Signal Source` to a value less than 1.0 (for example 0.99).

4.1.4 Variable Block

You may notice that the flow graph comes with a `Variable` block named "samp_rate." This block is a way to store constants or values common between several blocks and make it easy to modify the value for all the blocks using it. For example, the sample rate of the flowgraph is usually the same for all blocks, so it is a good idea to use a variable for such a value. We will now try using a variable to hold the musical note we want to play.

1. Add a `Variable` block to your flow graph.

2. Change the *ID* field to the name you want to reference it by. Use names that describe the value, like 'G' or 'G_note.'

3. Change the *Value* to 392.

4. Change the *Frequency* field of the `Signal Source` block to the name you gave your variable.

Now when you run the flowgraph, you should have a similar result. However, using variables in your flowgraphs make them much easier to understand and modify. You should always try to use variables when possible.

Question 1. When running the .wav file, you heard a constant tone corresponding to the note G. What happens to the sound if we change the *Waveform* field from a *Cosine* to a *Sine* wave? Why?

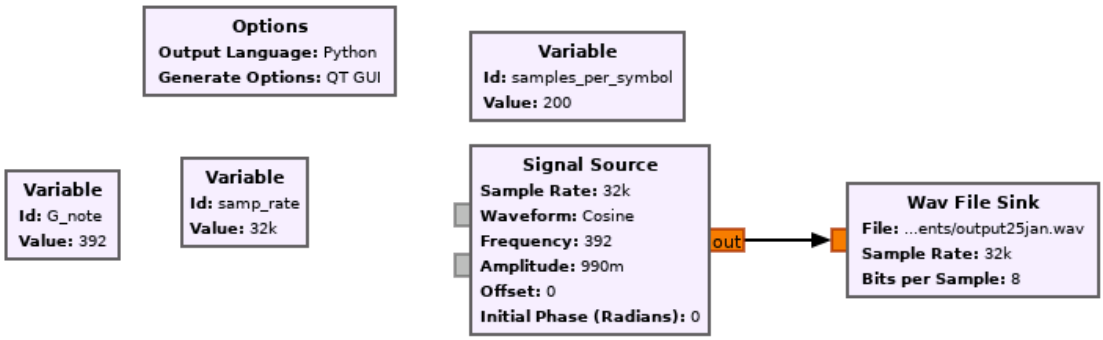

Figure 12: The final flowgraph of steps 3.1.

4.2 Add

Now that we can play one tone, let's try to combine two notes into a partial *chord*. In this case, we are going to combine the musical notes C and G. Listed below are the frequencies of each note.

Note	Frequency (Hz)
C	262
G	392

1. Delete your previous connection between the `Signal Source` and `Wav File Sink` by clicking on the arrow and pressing the 'Delete' button on your keyboard.
2. Create a `Variable` block for each musical note and name it appropriately.
3. Create another `Variable` block that contains the number of notes that our chord consists of and name it *num_notes*. In this case, it is two.
4. Add a `Signal Source` block for each note and be sure to set the frequency of each to a corresponding note variable. Also, don't forget to set the amplitude of each to $0.99V$.
5. Add an `Add` block to the flowgraph. Change its *IO Type* to Float and its *Num Inputs* to your variable holding the number of notes (*num_notes*).
6. Connect the `Signal Source` blocks to the inputs of the `Add` block.
7. Add a `Multiply Const` block and set the *Constant* to $1.0/num\_notes$.

8. Connect the output of the `Add` block to the `Multiply Const` block and then connect the `Multiply Const` to the `Audio Sink` block.

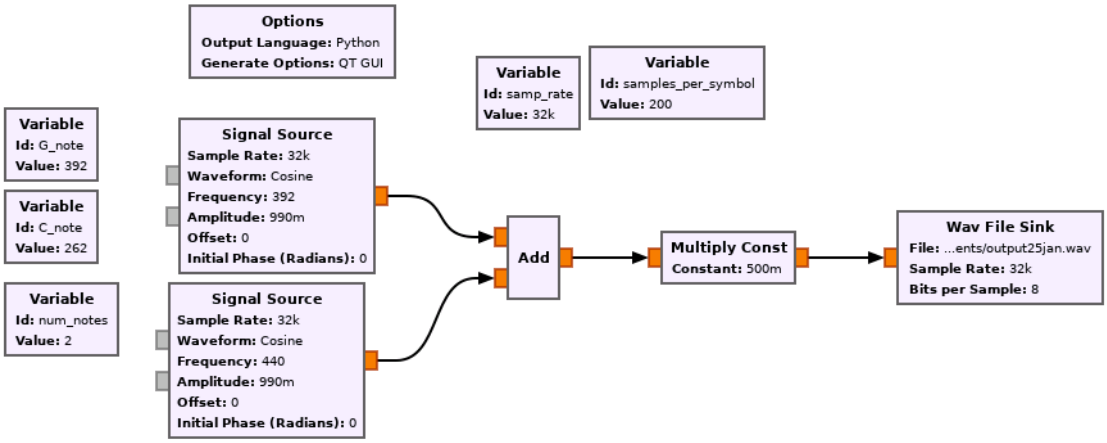

Figure 13: The final flowgraph of steps 3.2.

Question 2. What is the mathematical representation of the signal being generated? How many different notes are there in this tone? Format your answer as the sum of two sinusoidal signals (i.e, cosine or sine signals).

4.3 Multiply

Next, let's try multiplying the signals together instead of adding them.

1. Replace the `Add` block with a `Multiply` block
2. Set the *IO Type* to Float and the *Num Inputs* to your variable holding your number of notes
3. Connect the blocks the same as before.

When you run the flowgraph, you should notice a difference in the tone it produces.

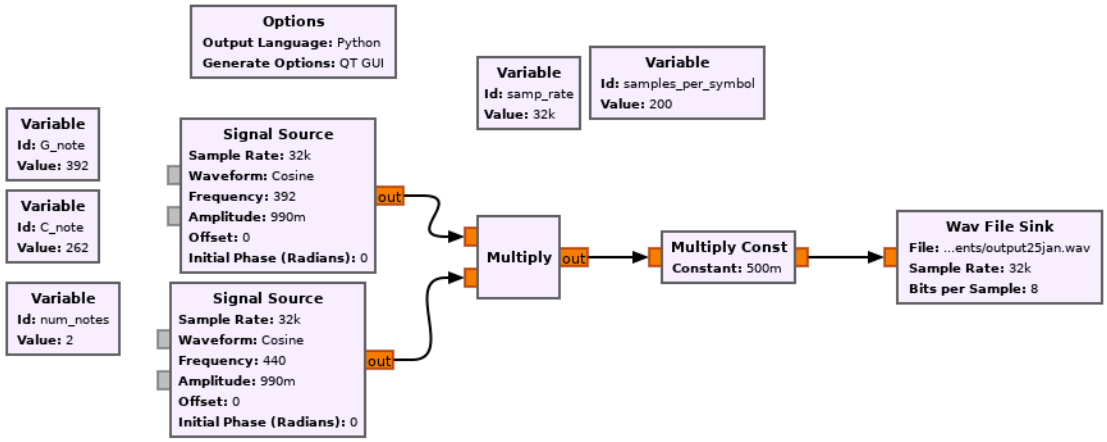

Figure 14: The final flowgraph of steps 3.3.

Question 3. How many different notes are produced? Are these notes (frequencies) higher or lower than when you added the blocks together? Justify your answer mathematically. Format your answer as the linear sum of two sinusoidal signals.

Hint: As an intermediate step to derive the answer, convert the sinusoidal representation of the signals into a different format.

4.4 Examining a Chord

Fig. 15 below provides the FFT flowgraph (i.e., frequency response) of a mystery *major chord*. For the definitions of a chord and major chord, see https://www.musictheory.net/lessons/40.

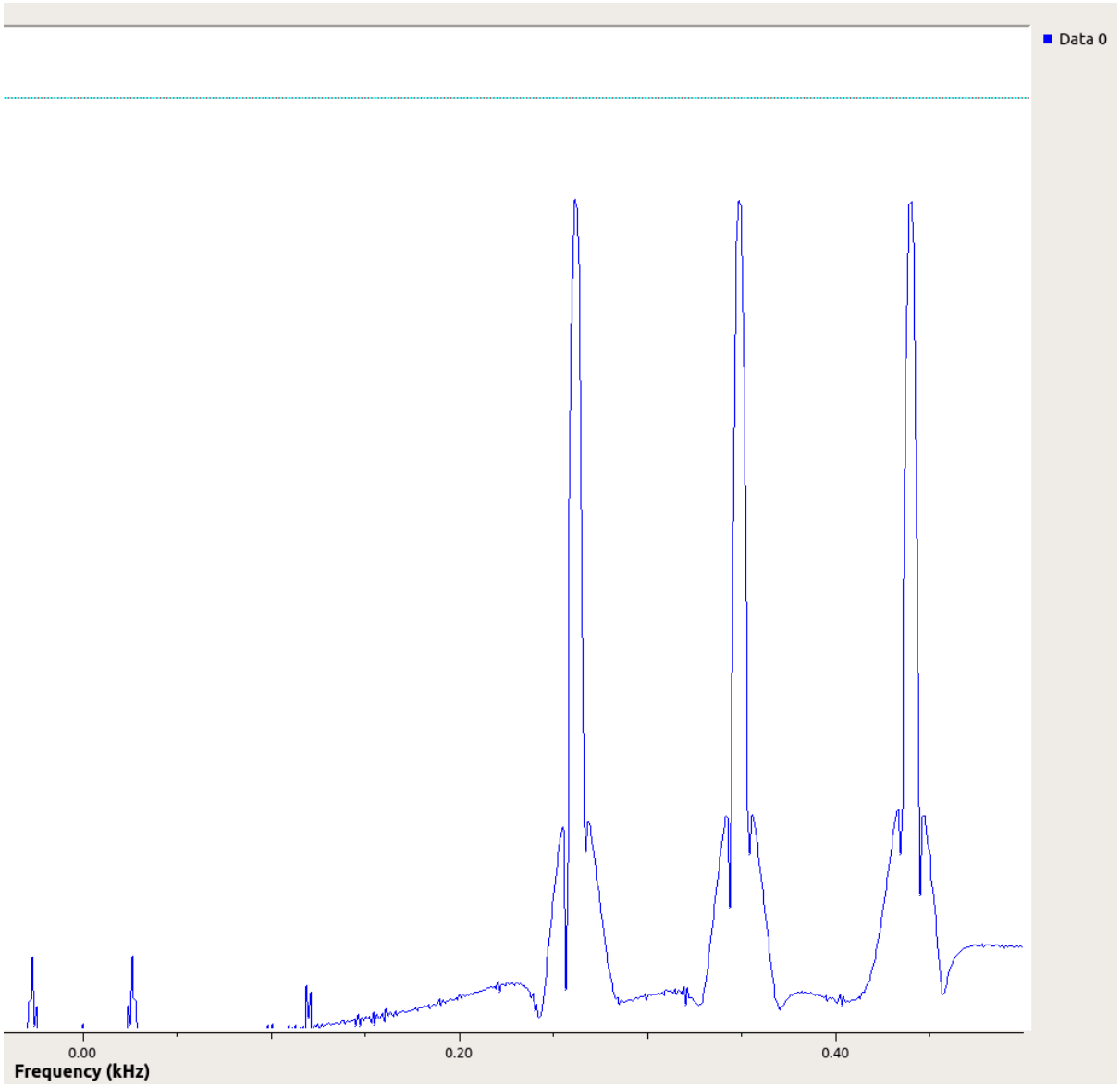

Figure 15: FFT of the Mystery Chord

The note frequencies you will need are listed below in Figure 16. Each column represents a specific *octave*.

Note	1	2	3	4	5	6	7	8
B	62	123	247	494	988	1976	3951	
A sharp/B flat	58	117	233	466	932	1865	3729	
A	55	110	220	440	880	1760	3520	
G sharp/A flat	52	104	208	415	831	1661	3322	
G	49	98	196	392	784	1568	3136	
F sharp/G flat	46	92	185	370	740	1480	2960	
F	44	87	175	349	698	1397	2794	
E	41	82	165	330	659	1319	2637	
D sharp/E flat	39	78	156	311	622	1245	2489	4978
D	37	73	147	294	587	1175	2349	4698
C sharp/D flat	35	69	139	277	554	1109	2217	4434
C	33	65	131	262	523	1047	2093	4186

Figure 16: Note Frequencies. Each column corresponds to a different octave.

Question 4. Three notes have been added together as seen in Figure 15. Which octave is this major chord in?

Question 5. Identify the notes and the major chord as seen in Figure 15. Below is a table of different major chords and their notes. Use this as a reference as the mystery chord is one of these possible combinations.

Chord	Notes
C	C, E, and G
E	E, G, and B
F	F, A, and C
G	G, B, and D

Question 6. Recreate the mystery major chord in GNU Radio. You can base your flowgraph on your answers from previous questions. Provide a screenshot of the flowgraph in your lab submission and include the grc file as an attachment.

To take a screenshot on the lab computers:

1. To open the screenshot utility, go to **Applications** to **Accessories** to **Screenshot**
2. Click on **Select area to grab** and drag the pointer over the **.grc** file to take a screenshot.

See the Figures below to find the screenshot application.

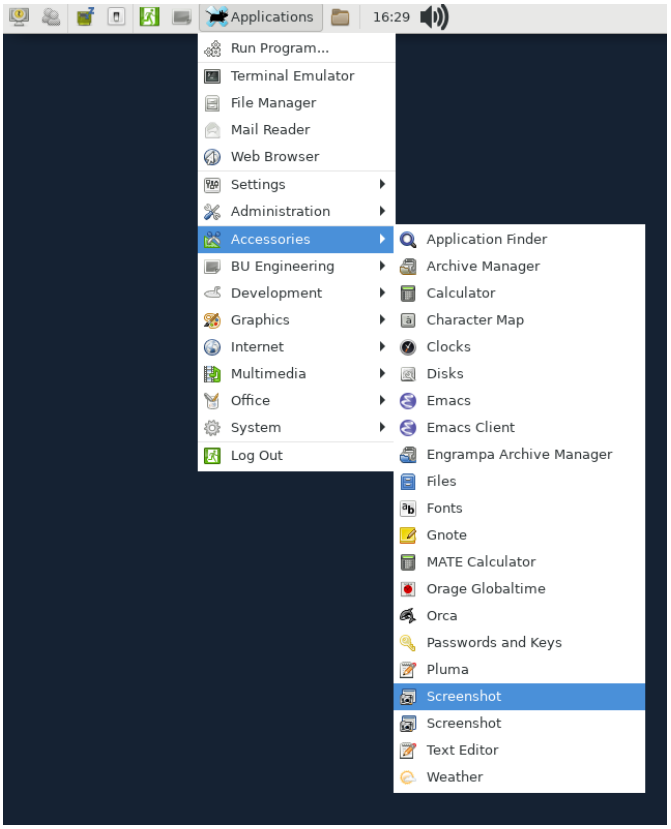

Figure 17: Path to Screenshot Application

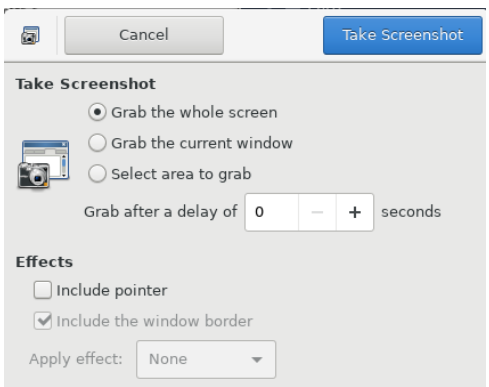

Figure 18: Screenshot Window