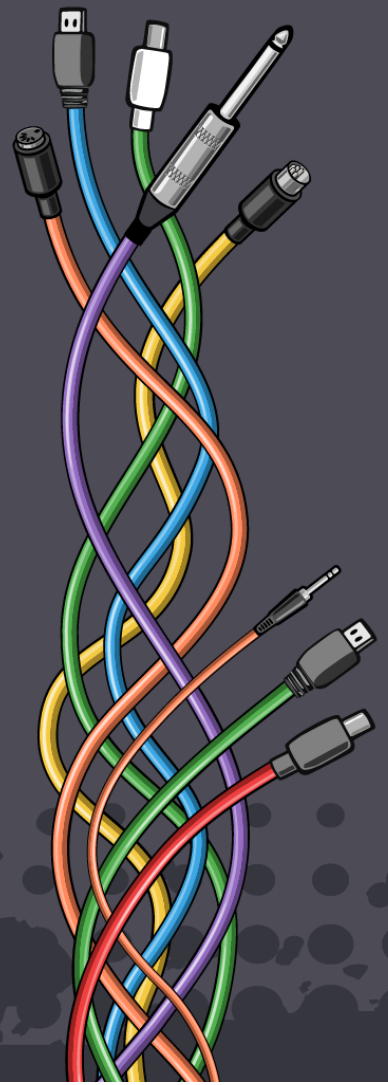


WORKSHOP:

FROM PAPER TO PLUGIN

ROSS CHISHOLM, JOEL ROSS
& JAMES HALLOWELL

ADC25 Bristol

Welcome

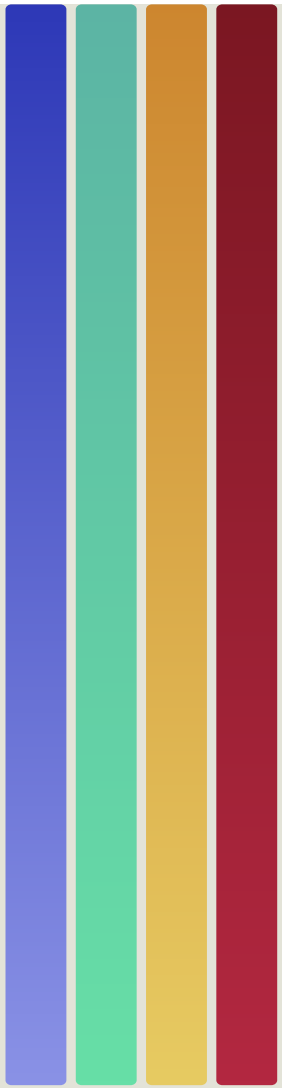
Focusrite novation

beating my head against that wall

“After I had been studying with him for two years, Schoenberg said, "In order to write music, you must have a feeling for harmony." I explained to him that I had no feeling for harmony. He then said that I would always encounter an obstacle, that it would be as though I came to a wall through which I could not pass. I said, 'In that case I will devote my life to beating my head against that wall.”

-John Cage

Overview



Focusrite novation

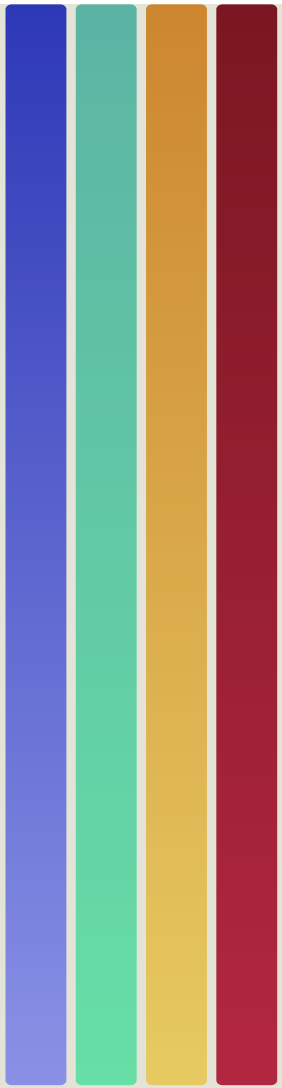
Outline

What are we going to talk about today?

- Introduction and a little history
- A look at the Paper
- Getting started with Jupyter Notebook
- An experiment with delays in Reaper
- Building a filter step-by-step
- Implementing the paper filter
- Porting to C++ and testing the port
- Bringing filters into a juce plugin
- Have some fun with it!

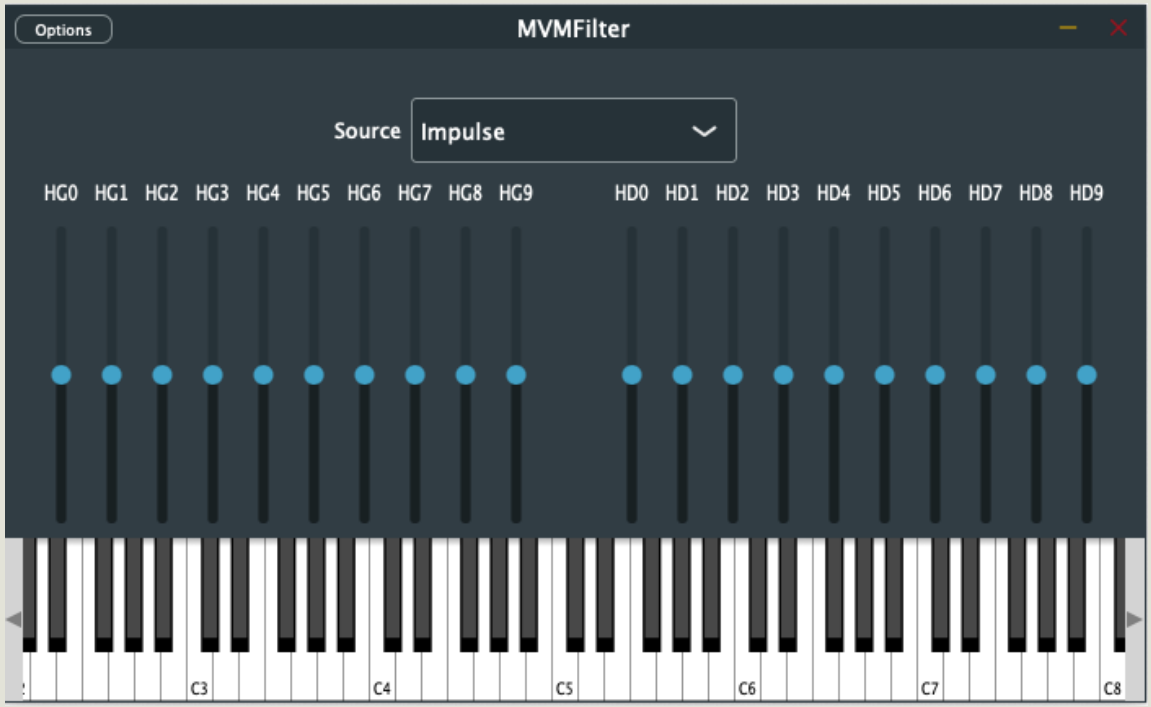
Focusrite novation

Introduction



Focusrite novation

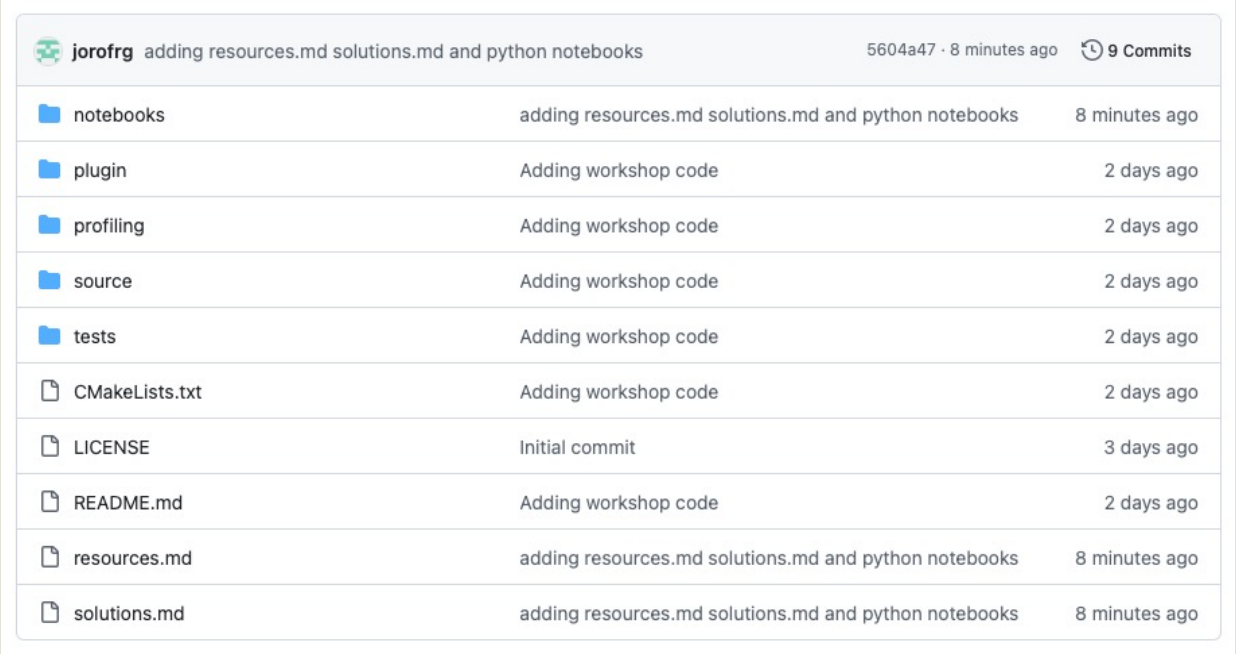
Plugin Demo



Focusrite novation

The paper-2-plugin repository

```
>git checkout workshop
```



jorofrg adding resources.md solutions.md and python notebooks — 5604a47 · 8 minutes ago — 9 Commits

notebooks	adding resources.md solutions.md and python notebooks	8 minutes ago
plugin	Adding workshop code	2 days ago
profiling	Adding workshop code	2 days ago
source	Adding workshop code	2 days ago
tests	Adding workshop code	2 days ago
CMakeLists.txt	Adding workshop code	2 days ago
LICENSE	Initial commit	3 days ago
README.md	Adding workshop code	2 days ago
resources.md	adding resources.md solutions.md and python notebooks	8 minutes ago
solutions.md	adding resources.md solutions.md and python notebooks	8 minutes ago

https://github.com/FocusriteGroup/paper-2-plugin-adc25/

Focusrite novation

A little history

We are going to look at a paper by Max Mathews and Julius O. Smith III

If you haven't heard these two names…

Max was responsible for the earliest computer music, beginning in the 1950s

Max/MSP was named after Max and almost all of the 'music programming languages' and patching systems that exist today owe a debt to the design of Max's MUSIC-N system. His work also inspired the singing HAL-9000 in 2001: A Space Odyssey.

Julius O. Smith, building on work with John Chowning (responsible for the FM synthesis in the DX-7 synthesizer), worked with Yamaha on waveguide synthesis for the VL-1 and contributed to the design of Shazam.

Max Mathews

Julius O. Smith III

Introducing the Paper

Focusrite novation

Methods for Synthesizing Very High Q Parametrically Well Behaved Two Pole Filters

Max Mathews
Julius O. Smith III

Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, California 94305

Stockholm Musical Acoustic Conference (SMAC)

August 6-9, 2003

Abstract

Techniques for synthesizing two pole filters are well known. A number of techniques introduce unpleasant sounding transients in the filter response when the frequency or damping of the filter is rapidly changed. We will demonstrate a difference equation for a digital filter in which both the frequency and the damping can be changed without producing discontinuities in the filter output. The technique is based on the well known property of the product of complex numbers. In polar form, the magnitude of the product of two numbers is the product of their magnitudes and the angle of the product is the sum of their angles. Successive multiplies can produce a rotating vector whose real or imaginary parts are samples of constant amplitude sine waves or of exponentially damped sine waves. The frequency and damping of the resulting waves can be changed without changing the amplitude of the waves. These properties can be used to make a digital filter whose input, frequency, and damping can all be functions of time in a useful way. Two alternative structures are additionally proposed, having better numerical properties for low-cost fixed-point implementations. A program to demonstrate some musical applications of these filters will be shown.


DON'T
PANIC!
Focusrite
novation

All we need!

$$x(n+1) = x_1 x(n) - y_1 y(n) + u(n) \quad (12)$$
$$y(n+1) = y_1 x(n) + x_1 y(n) \quad (13)$$

$$x_1 = e^{-\frac{1}{\tau f_s}} \cos\left(2\pi \frac{f}{f_s}\right) \quad (7)$$
$$y_1 = e^{-\frac{1}{\tau f_s}} \sin\left(2\pi \frac{f}{f_s}\right) \quad (8)$$

$$\tau \triangleq \text{time in seconds for filter to decay to } 1/e \quad (9)$$
$$f \triangleq \text{resonant frequency of filter in Hz} \quad (10)$$
$$f_s \triangleq \text{sampling rate in samples per second} \quad (11)$$

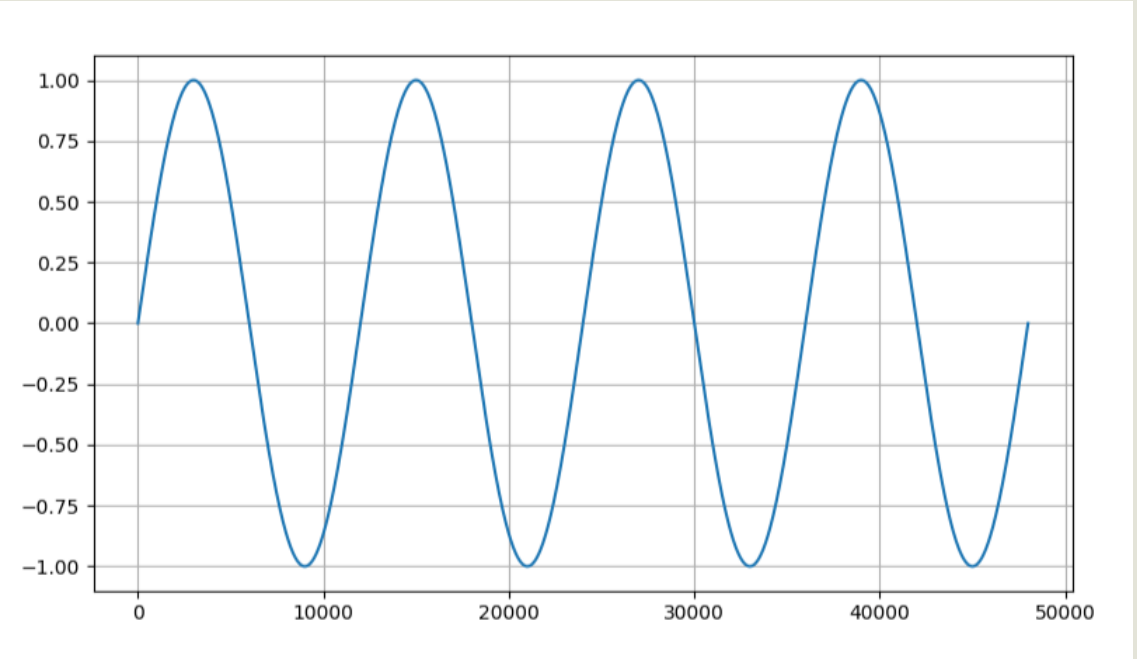
https://en.wikipedia.org/wiki/RC_time_constant

Jupyter Notebook, Warm Up

Focusrite novation

Intro to Jupyter Notebook, numpy and matplotlib

In order to get started, lets try to do something simple in jupyter notebook that shows a plot of a sine wave…



Sine Wave Plot

Set up

If you've followed the instructions in the readme, you should have python 3 installed, and a virtual environment set up.

1. Follow the instructions to activate the virtual environment on your platform.
2. run `jupyter notebook --no-browser` and open the notebook.
3. create a new notebook and give it a name.

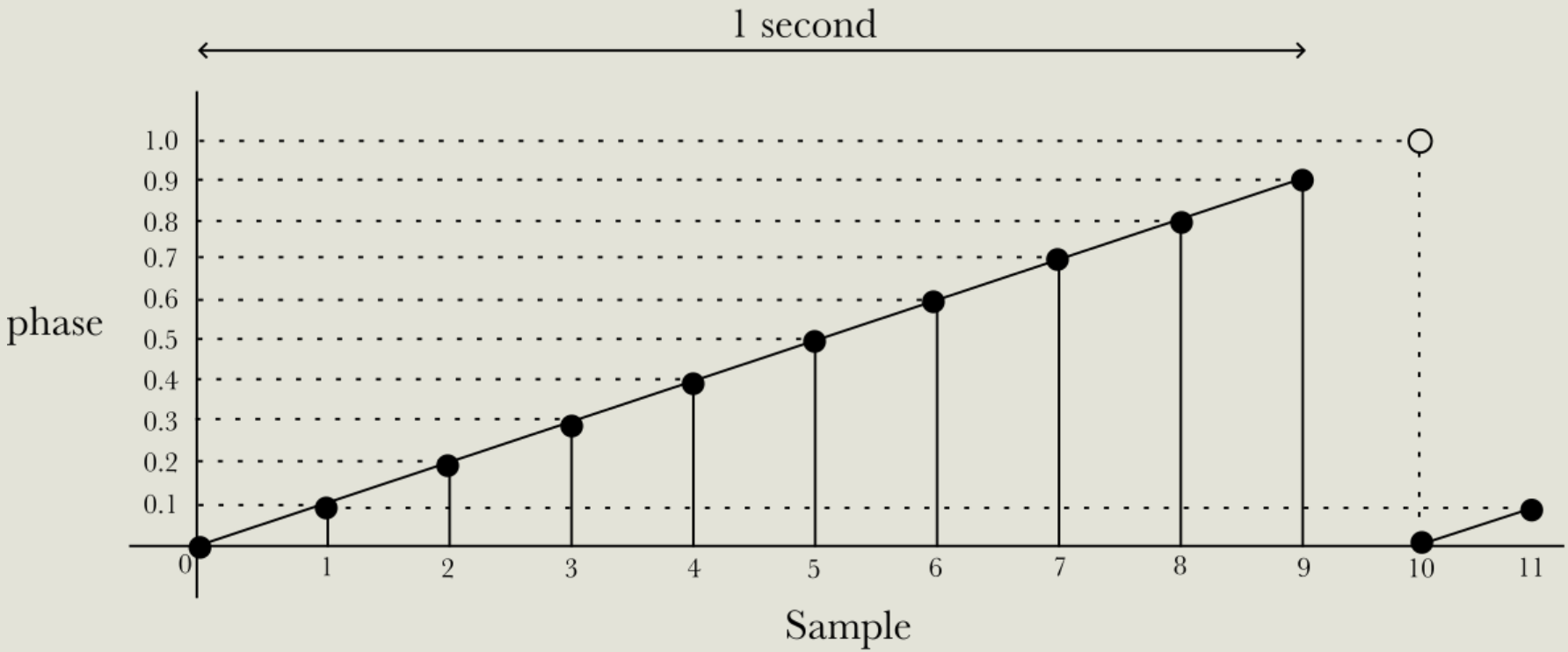
We need to load the `numpy` and `matplotlib` libraries so create a new cell by clicking and add.

```
import numpy as np
import matplotlib.pyplot as plt
```

Defining the sine

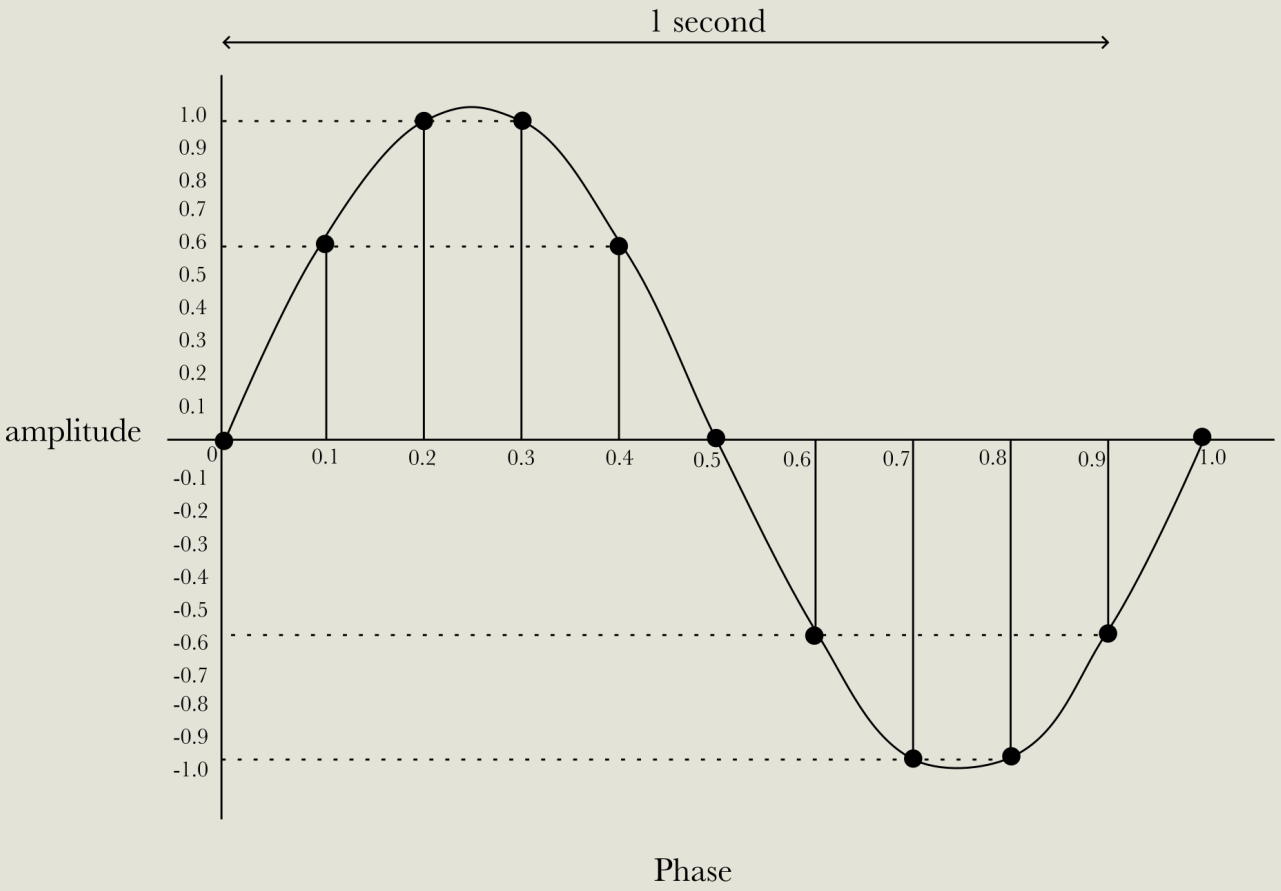
```
# Draw a sine wave given
# - a sample rate
# - a frequency
# - a duration

sample_rate = 48000 # samples/second
frequency = 4.0 # Hz
duration = 1.0 # Seconds

# the number of samples will be sample_rate * duration
N = int(np.ceil(sample_rate * duration))
```

1 cycle-per-second phase at 10 samples-per-second



Focusrite novation

1 cycle phase normalised Sine at 10 samples-per-second



Focusrite novation

Back to the sine

To give a particular frequency, we can work out what the phase difference is every sample…

```
# The phase increment every sample is going to be
increment = frequency / sample_rate
```

This means every sample, we're going to move a step the size of `increment` along the x-axis.

Also remember that a single cycle of the sin or cos function spans 2π, so we need to scale this by `2 * np.pi`.

```
for ndx in range(0, N):
    signal[ndx] = np.cos(2 * np.pi * increment * ndx)
```

(Note that multiplying `increment` by `ndx` is the same as repeatedly adding up increment because ndx=1,2,3,…)

Sampling and the Nyquist

There is a frequency referred to as the Nyquist frequency which is half the sampling frequency and which is the highest frequency we can represent for a given sample rate.

An intuition into why this is can be gained by thinking about what happens when you sample a cosine wave at the Nyquist frequency.

Try playing with the plotting parameters so you can see what happens at

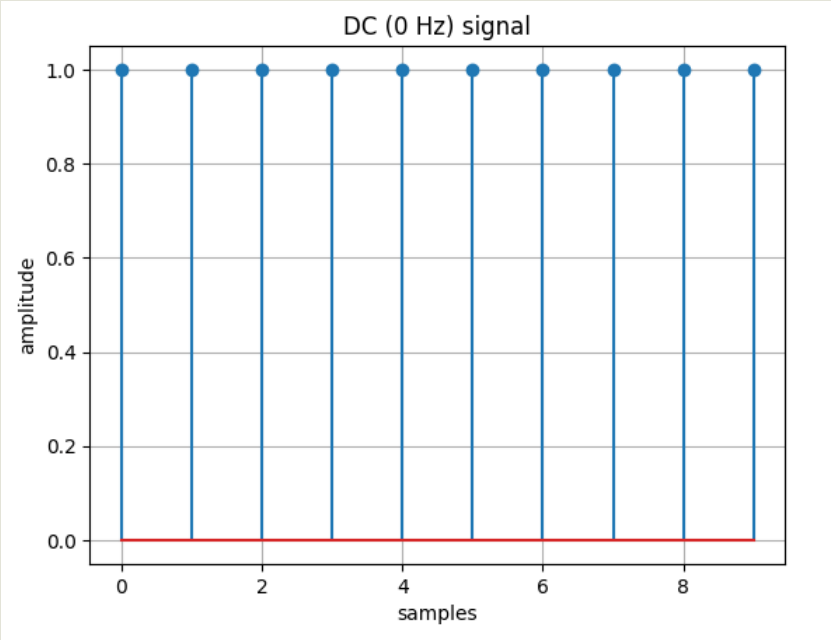
Half the Nyquist frequency

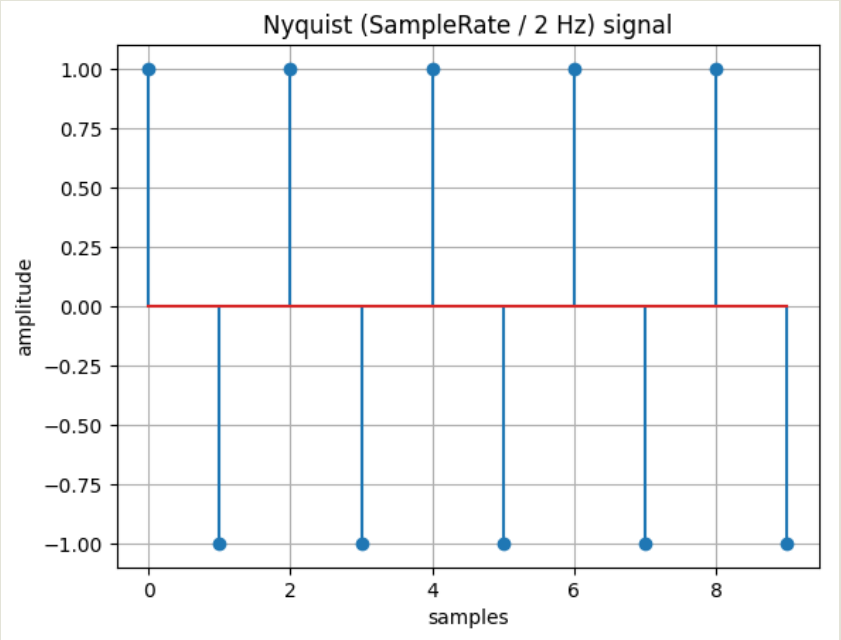
The Nyquist frequency

```
sample_rate = 1000 # samples/second
frequency = 500 # Hz
duration = 0.01 # Seconds
```

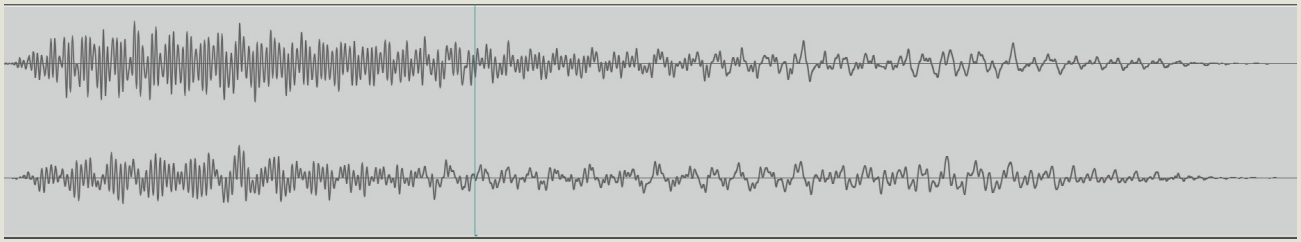
(It can be helpful to replace `np.plot` with `np.stem`)

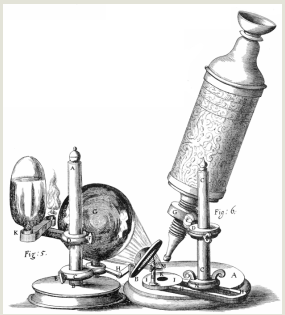
Two important signals



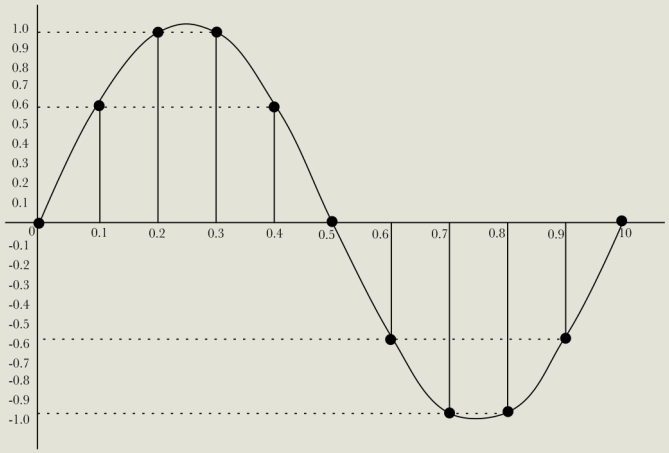


Frequency Decomposition





Credit: Robert Hooke, Micrographia



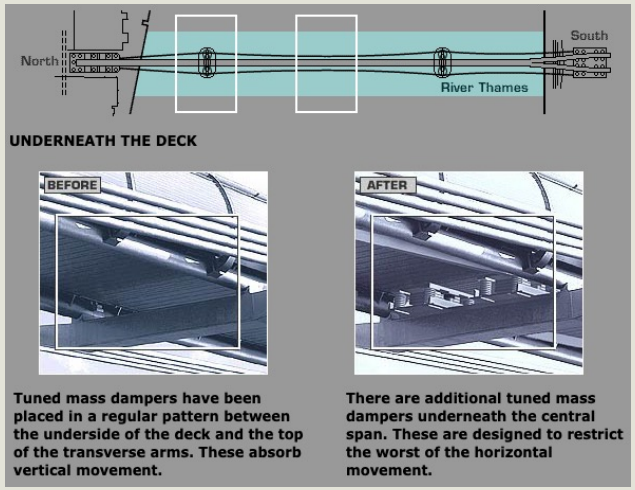
Building a general filter function

Focusrite novation

What is a filter anyway?

When we talk about filters generally, we might mean anything which takes an input and does something to it and gives you back this modified thing. In this view, anything that transforms sound is a filter.

Your guitar amp is a filter, the cabinet too and every guitar pedal in your rack.



http://news.bbc.co.uk/hi/english/static/in_depth/uk/2000/millennium_bridge/plan_under_deck.stm

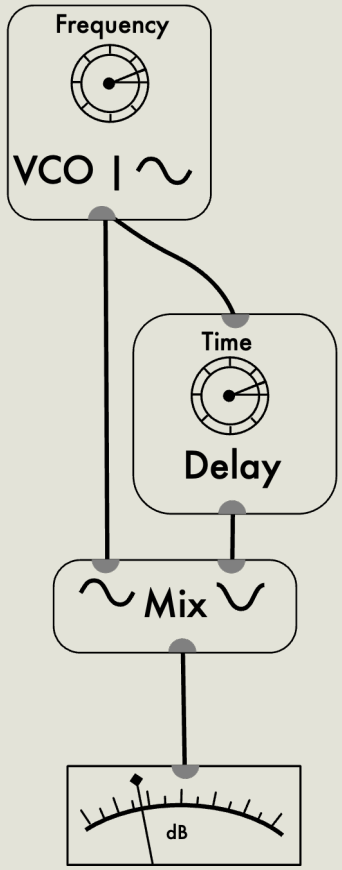
A little experiment

Focusrite novation

A Patching diagram

We are going to try a little experiment where we set up a patch using this patching diagram. If you have Max/MSP or Pure Data and are confident to set this up, then go ahead and do it that way. Otherwise, instructions for doing this in reaper with built-in plugins below.

In the block diagram we have a tone generator `VCO` with controllable frequency, a time delay `Delay` with controllable time, a mixer and a power meter.



Method

Open Reaper and turn the output bus volume down to −∞.

Create a new track and add the `Tone Generator Plugin`

Add the `ReaDelay` plugin after that

Add a `VU Meter` below `ReaDelay`

Go to `ReaDelay`

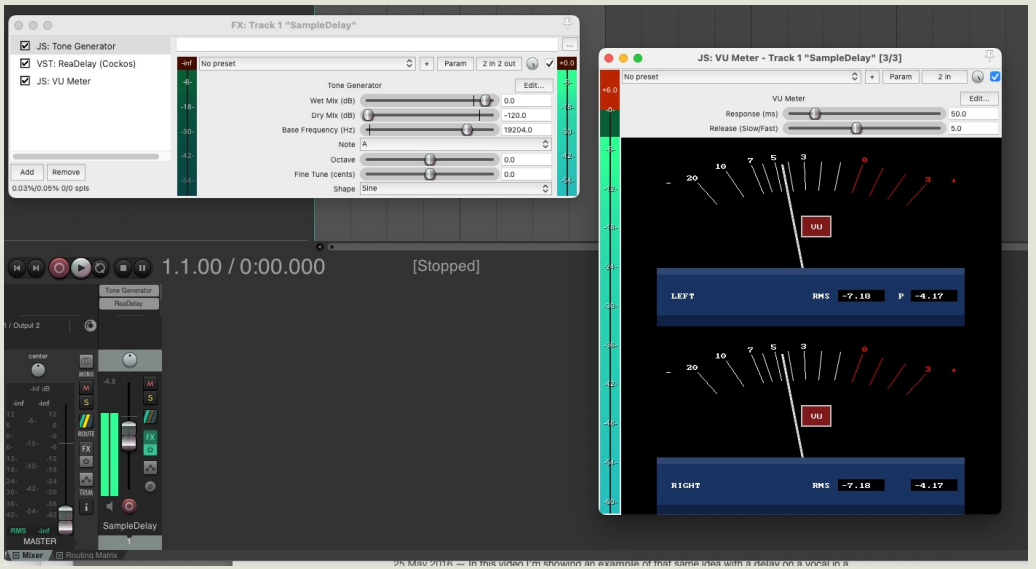
- Set the Wet/Dry mix to 100% Wet + 100% Dry
- Set the `Length (musical)` slider to **0**
- Double click the `Length (time)` number-box and enter 0.02 (don't worry that it looks like it resets to 0, it has really set 0.02 ms!)

Now go to the `Tone Generator`

- sweep the frequency around and watch the VU meter

Try playing with the delay time and see how the result changes.

How many 'dips' are there in the output where the power goes below -20dB?



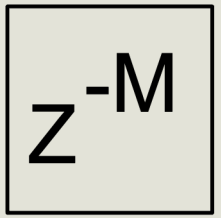
Reaper set up

Focusrite novation

Building Blocks

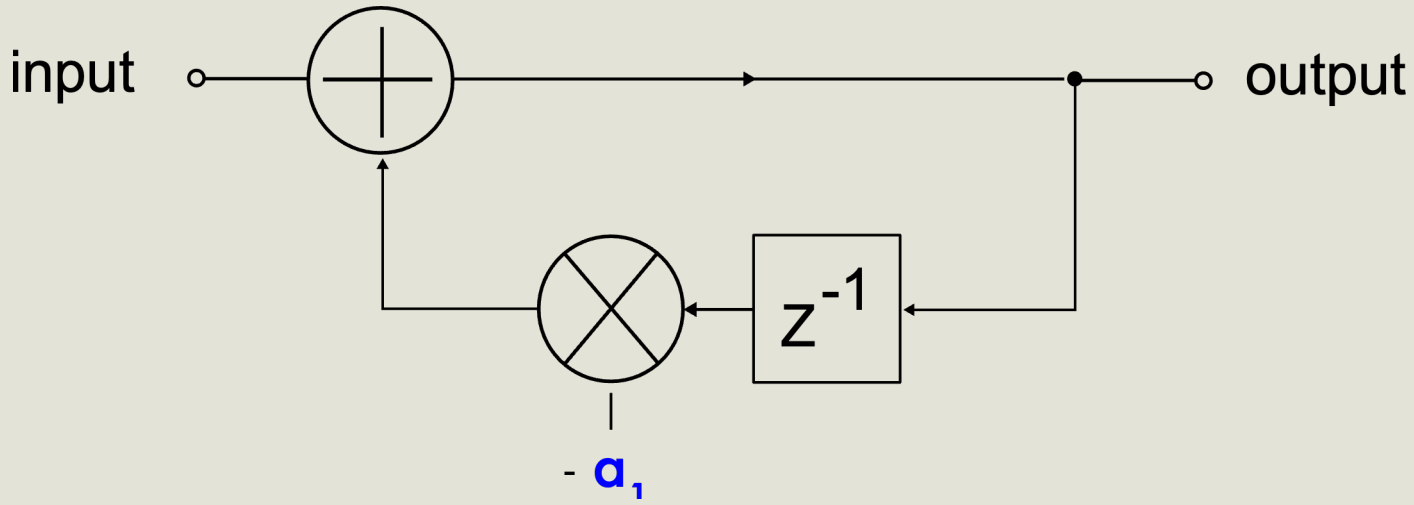
Multiply

Add



Delay

Block Diagram

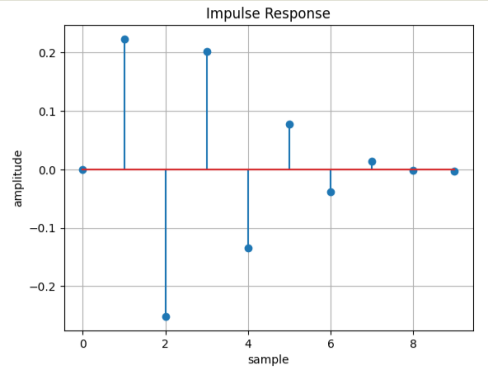


Tools

Difference Equation

$$y_n = x_n - a_1 y_{n-1}$$

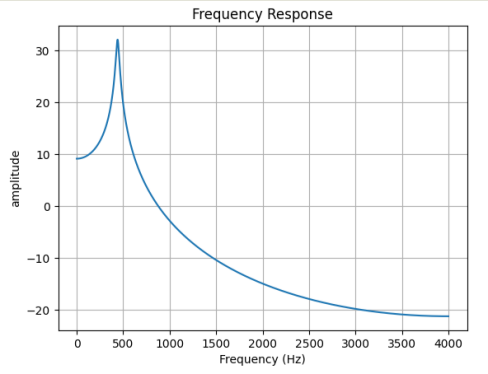
Impulse Response



$$H(z) = \frac{1}{1 + a_1 z^{-1}}$$

Transfer Function

Frequency Response



Focusrite novation

Filters 1

Difference Equation

$$y_n = x_n$$

Transfer Function

Our input is $X(z)$ - this is our input as seen through our microscope at frequency z.

$Y(z)$ represents our output in this world.

$$Y(z) = X(z) \cdot H(z)$$

$$H(z) = \frac{\text{output}}{\text{input}} = \frac{Y(z)}{X(z)} = 1$$

Code

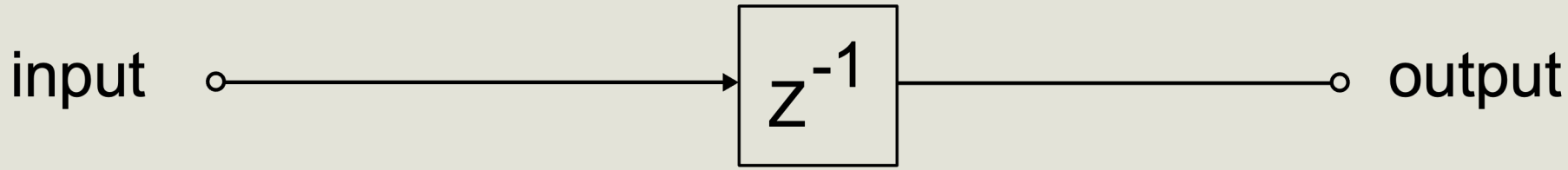
```
y = x
```

```
def filterfunction(input):
    return input
```

```
class FilterClass:

    def process(self, inputValue):
        outputValue = inputValue
        return outputValue
```

Focusrite novation

Filters 2



Difference Equation

$$y_n = x_{n-1}$$

Transfer Function

z raised to an index is an operation that shifts our signal in time by a number of samples equal to the index so we can use the substitution.

Difference Equation	Transfer Function
y_n	$Y(z)$
x_n	$X(z)$
x_{n-1}	$X(z)z^{-1}$
x_{n-2}	$X(z)z^{-2}$

Transfer Function

$$\text{output} = \text{input} \times z^{-1}$$

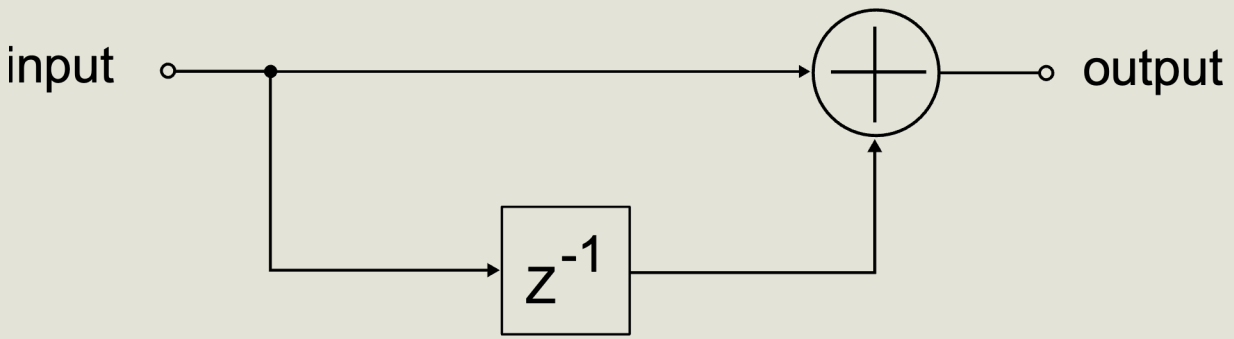
$$y_n = x_{n-1}$$

$$\begin{aligned} Y(z) &= X(z)z^{-1} \\ \Rightarrow \frac{Y(z)}{X(z)} &= z^{-1} \\ \Rightarrow H(z) &= z^{-1} \end{aligned}$$

Code

```
class FilterClass:

    def __init__(self):
        self.state = 0

    def process(self, inputValue):
        outputValue = self.state
        self.state = inputValue
        return outputValue

zm1 = FilterClass()
print(zm1.process(1.0))
print(zm1.process(0.0))
```

Filters 3



Difference Equation

$$y_n = x_n + x_{n-1}$$

Transfer Function

$$H(z) = 1 + z^{-1}$$

Transfer Function

Difference Equation	Transfer Function
y_n	$Y(z)$
x_n	$X(z)$
x_{n-1}	$X(z)z^{-1}$
x_{n-2}	$X(z)z^{-2}$

Now if we look at the difference equation

$$y_n = x_n + x_{n-1}$$

we now use the table, write

$$Y(z) = X(z) + X(z)z^{-1}$$

We can simplify this a bit by collecting the $X(z)$'s

$$Y(z) = X(z)(1 + z^{-1})$$

Then remembering that our transfer function is $H(z) = \frac{Y(z)}{X(z)}$,

rearrange so we have

$\frac{Y(z)}{X(z)} = 1 + z^{-1}$, so $H(z) = 1 + z^{-1}$

Using the Transfer Function

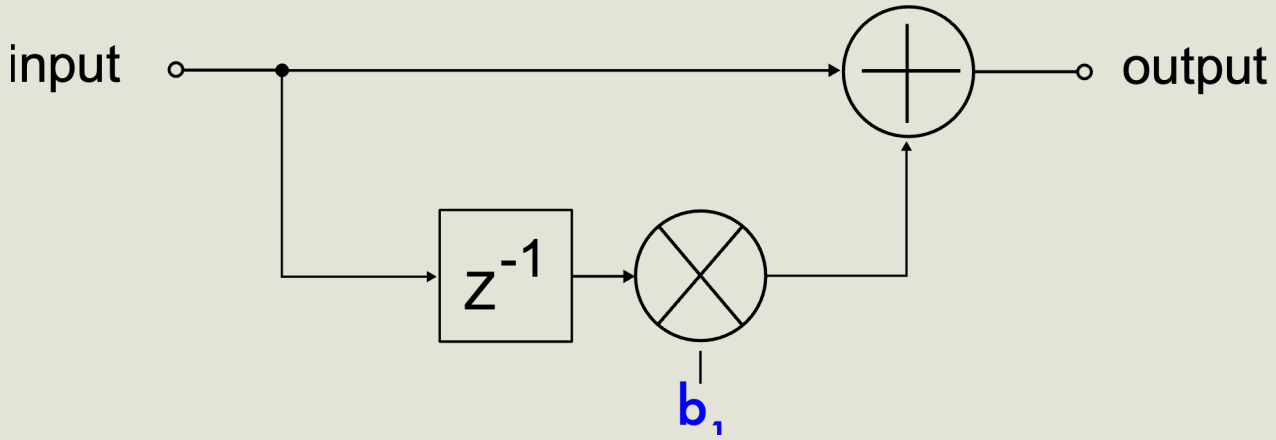
Taking the magnitude of the Transfer Function tells us what the Frequency response is at a frequency z.

$$|H(z)| = |1 + z^{-1}|$$

We can ask the question in reverse: at what frequency z is the response 0.

$$\begin{aligned} |H(z)| = |1 + z^{-1}| &= 0 \\ \rightarrow |z + 1| &= 0 \\ \rightarrow z &= -1 \end{aligned}$$

Filters 4



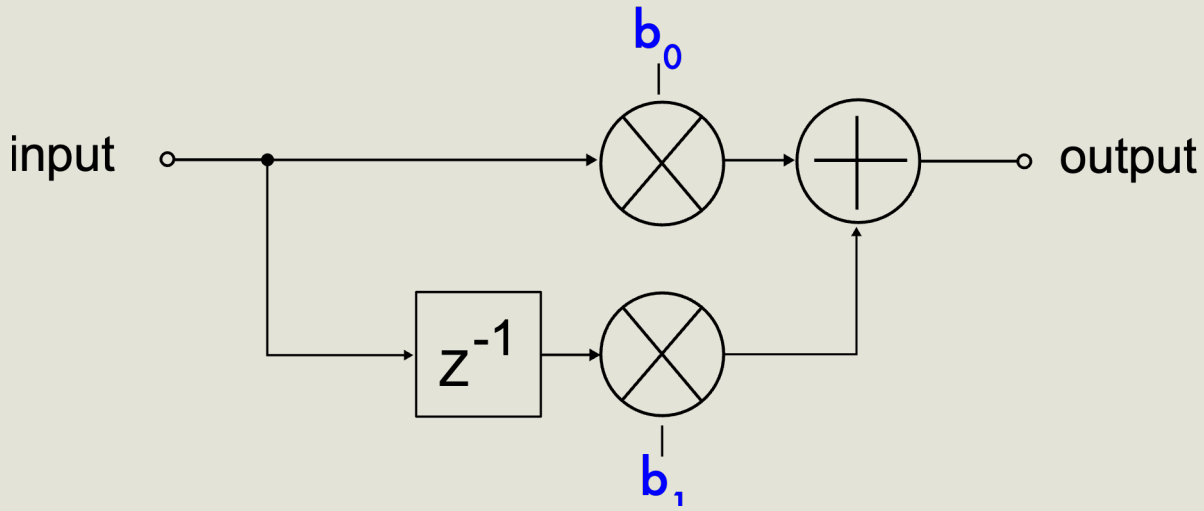
Difference Equation

$$y_n = x_n + b_1 x_{n-1}$$

Transfer Function

$$H(z) = 1 + b_1 z^{-1}$$

Filters 5



Difference Equation

$$y_n = b_0 x_n + b_1 x_{n-1}$$

Transfer Function

$$H(z) = b_0 + b_1 z^{-1}$$

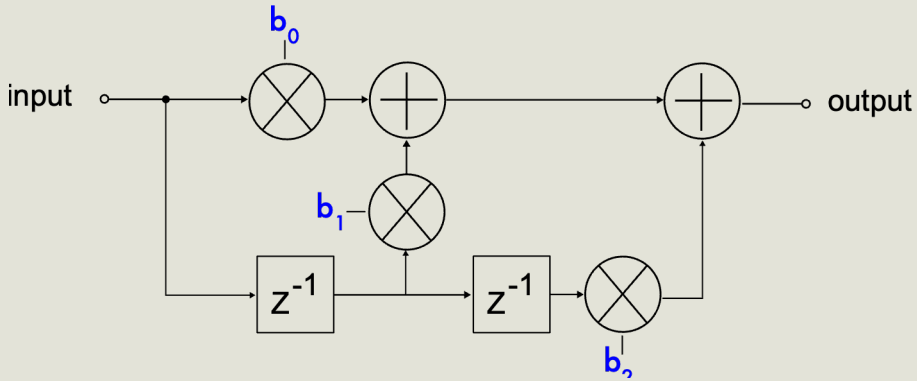
Filters 6

Difference Equation

$$y_n = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2}$$

Transfer Function

$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2}$$



Second Order Feed Forward

15 Minute Break

Focusrite novation

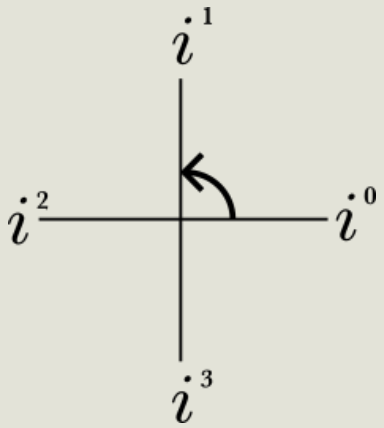
Complex Numbers refresher

Focusrite novation

The imaginary unit

We introduce the imaginary unit i with the rule $i^2 = -1, \quad i \neq 1$.

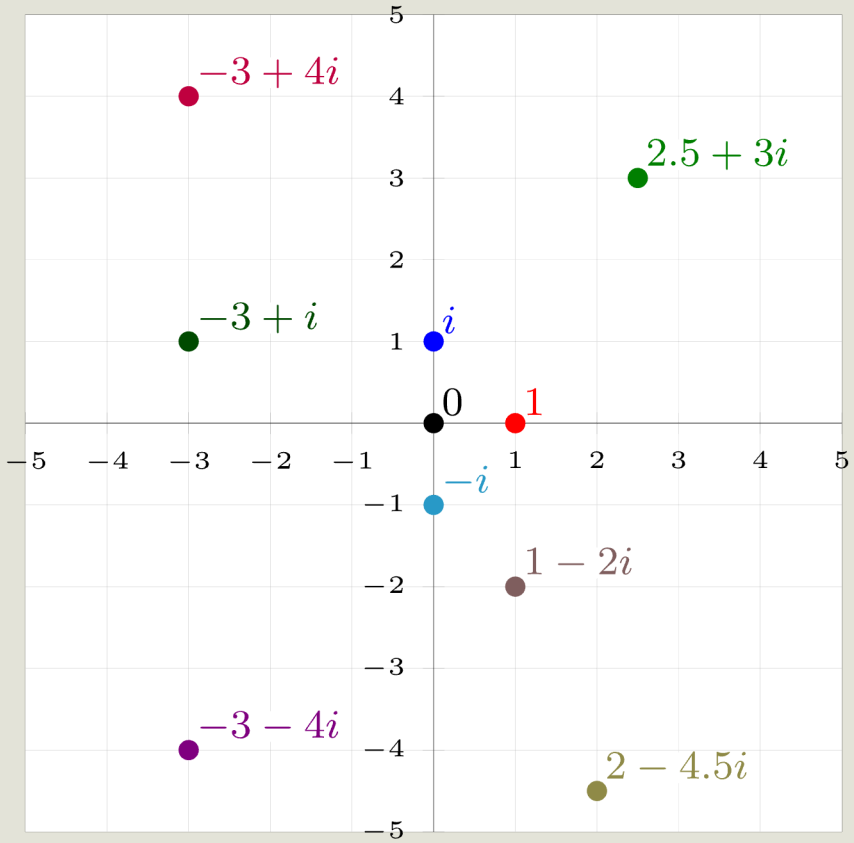
$$1 = i^0$$
$$1 \times i = i = i^1$$
$$i \times i = -1 = i^2$$
$$-1 \times i = -i = i^3$$
$$-i \times i = 1 = i^0$$



As Coordinates

We can treat these complex values as coordinates into a plane called the *complex plane*.

```
struct Complex {
    float real;
    float imaginary;
};
```



Complex Numbers in the plane

Illustration by Fschwarzentruber - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=132178373

Focusrite novation

Complex Values as angles with magnitude

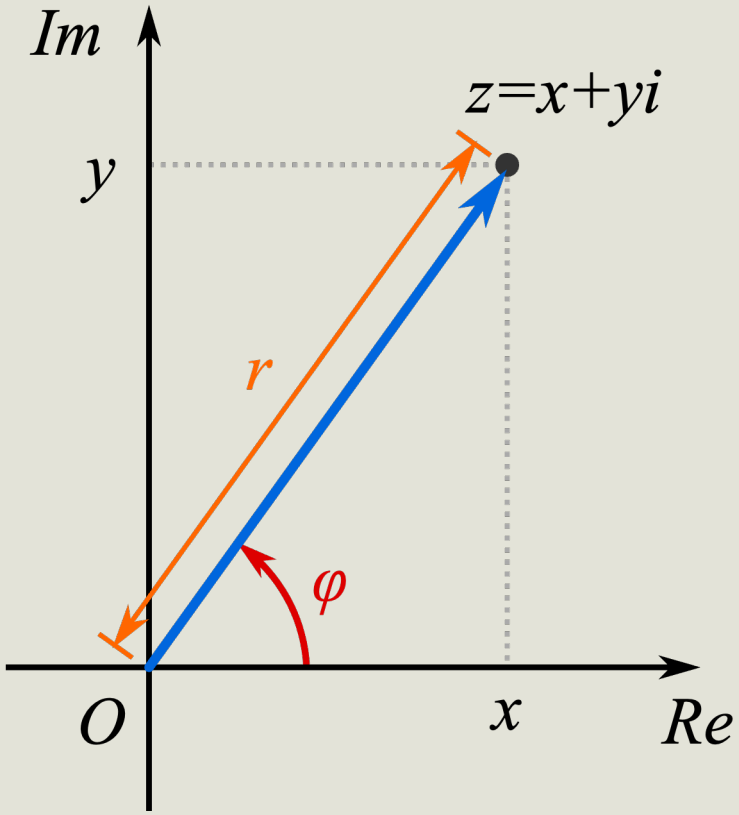
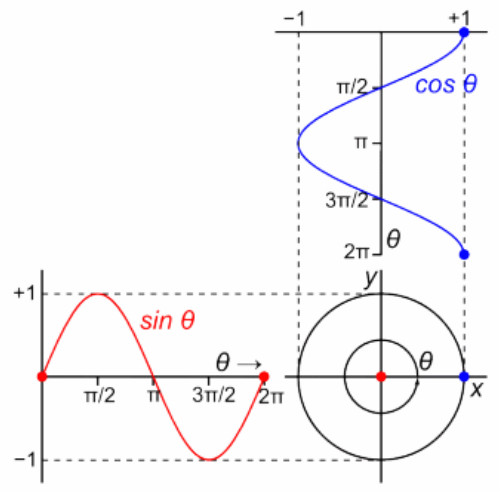


Illustration by Kan8eDie - Own work based on: Complex number illustration.svg by Wolfkeeper, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=5922759

Focusrite novation

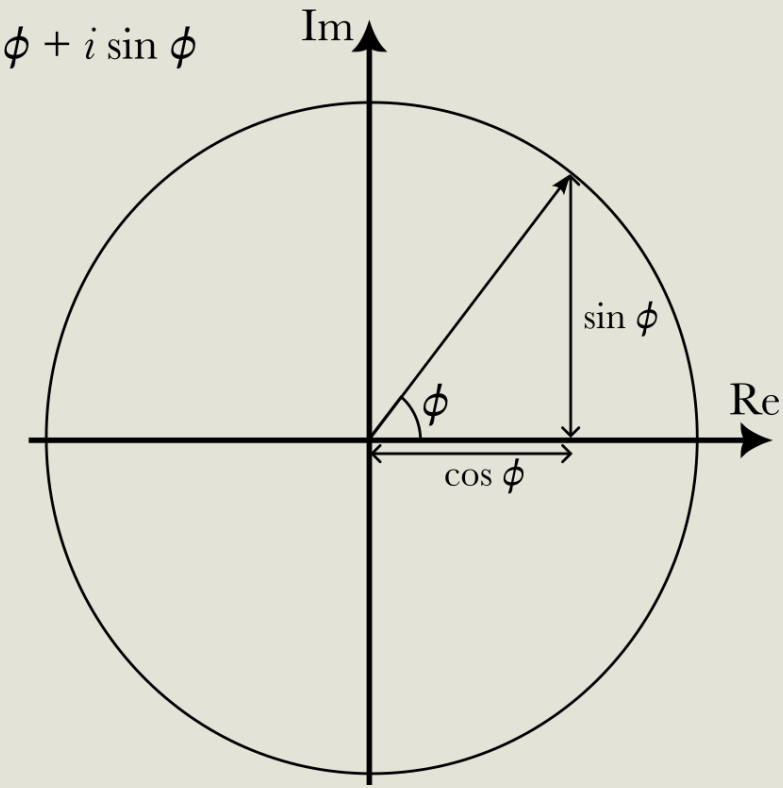
Euler's Formula

$$Ae^{i\phi} = A\cos\phi + A\sin\phi i$$



Credit: Chetvorno, CC0, via Wikimedia Commons

$$e^{i\phi} = \cos\phi + i\sin\phi$$



Focusrite novation

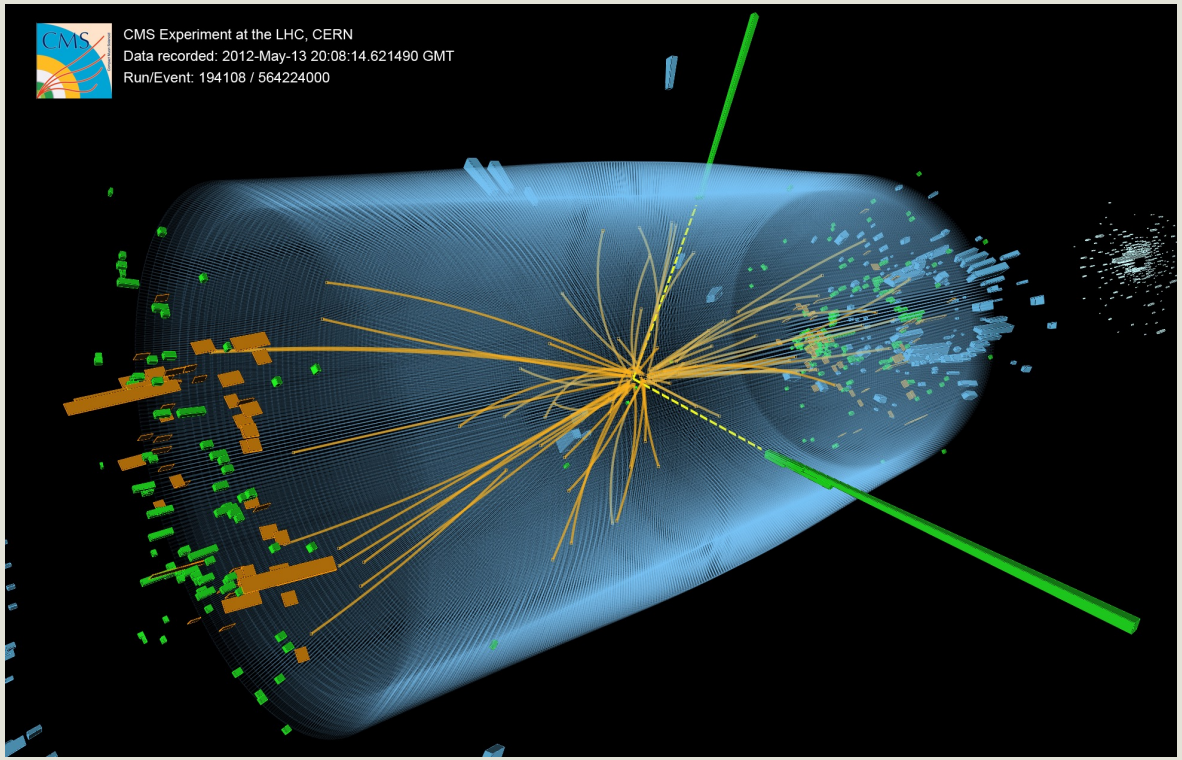
Frequency as rotation

We can also use the complex exponential to specify a frequency ω in radians-per-sample. That means that with every sample tick that passes, the value rotates by ω radians around the circle. So if we specify omega and find the value $e^{i\omega}$.

```
omega = 2 * (np.pi / 8)
rotation = np.exp(1j * omega)
```

If we start at $1 + 0\ i$ and keep multiplying it by the value of `rotation`, then after 8 steps we will have gone around the whole circle and returned to $1 + 0\ i$

Splitting the Atom



Credit: Cern

Z as frequency

…or splitting the atom

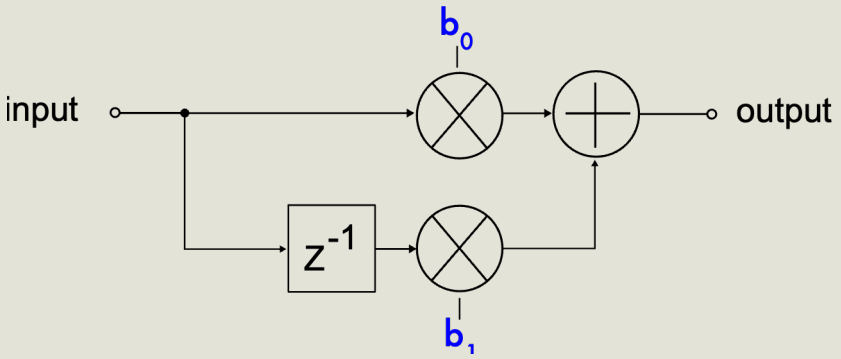
$$H(z) = b_0 + b_1 z^{-1}$$

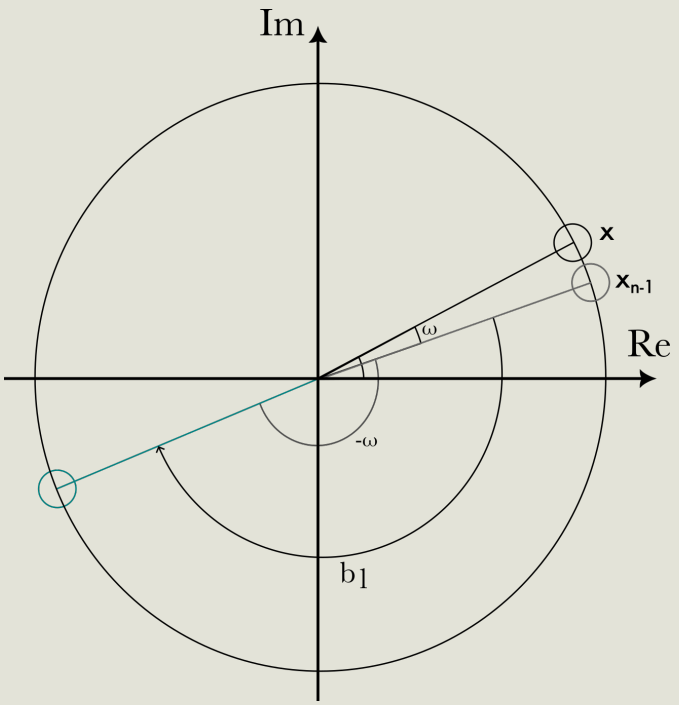
$$|H(z)| = |1 + b_1 z^{-1}| = 0$$
$$\longrightarrow |z + b_1| = 0$$
$$\longrightarrow z = -b_1$$

So to achieve a notch at a particular frequency ω we can choose z and set

$$b_1 = -z$$

$$z = e^{i\omega}$$





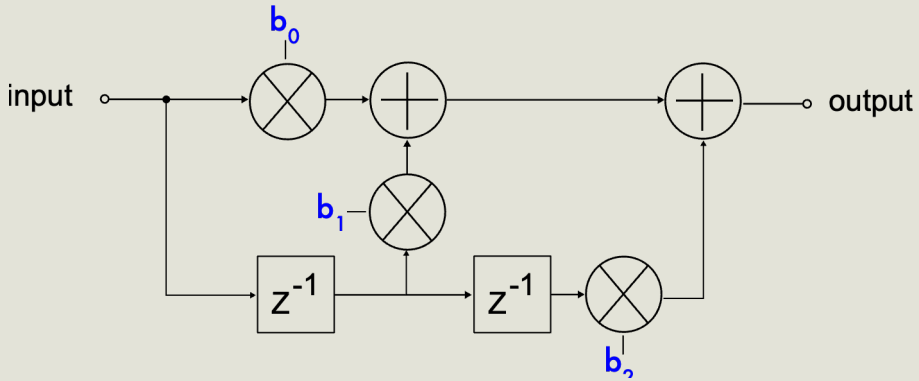
Focusrite novation

The 2nd order filter revisited

…putting it back together again

Transfer Function

$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2}$$



$$G(z) = z + \mathrm{b}_\omega$$

$$H(z) = |z + \mathrm{b}_\omega| \cdot |z + \overline{\mathrm{b}_\omega}|$$

$$H(z) = z^2 + \overline{\mathrm{b}_\omega} z + \mathrm{b}_\omega z + |\mathrm{b}_\omega|^2$$

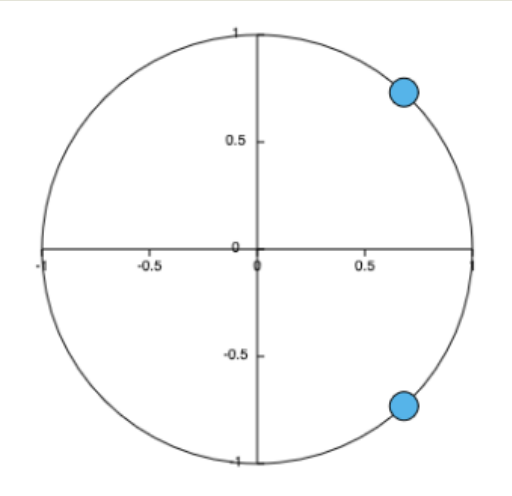
$$H(z) = z^2 + (\overline{\mathrm{b}_\omega} + \mathrm{b}_\omega) z + |\mathrm{b}_\omega|^2$$

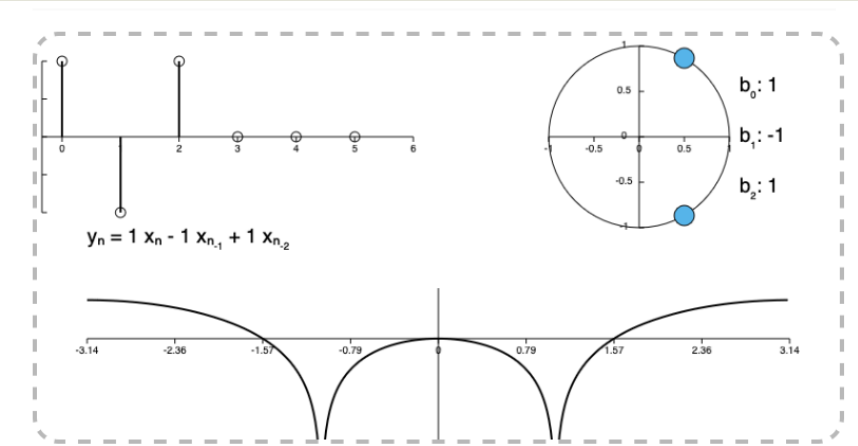
$$H(z) = z^2 + 2\mathrm{Re}\{\mathrm{b}_\omega\} z + |\mathrm{b}_\omega|^2$$

$$H(z) = 1 + 2\mathrm{Re}\{\mathrm{b}_\omega\} z^{-1} + |\mathrm{b}_\omega|^2 z^{-2}$$

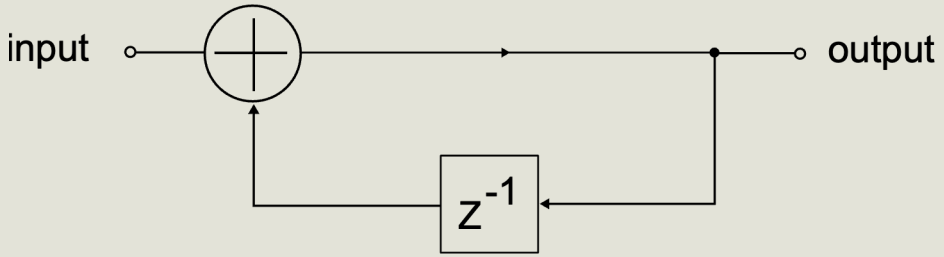
Zeros of a filter

A new analysis tool





Filters 7



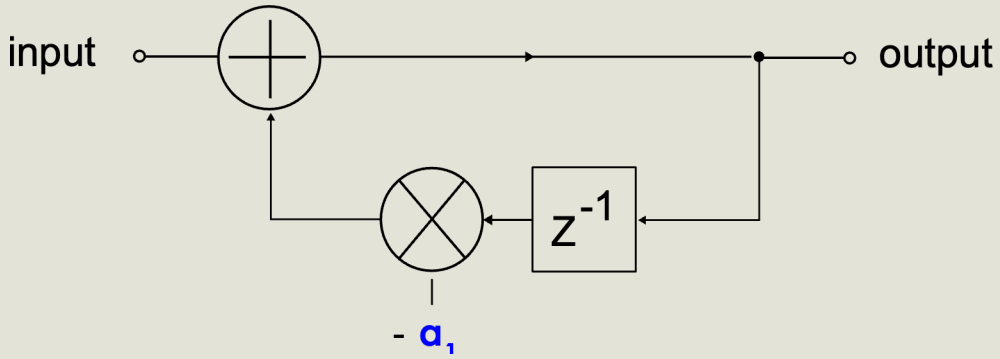
Difference Equation

$$y_n = x_n - y_{n-1}$$

Transfer Function

$$H(z) = \frac{1}{1 + z^{-1}}$$

Filters 8



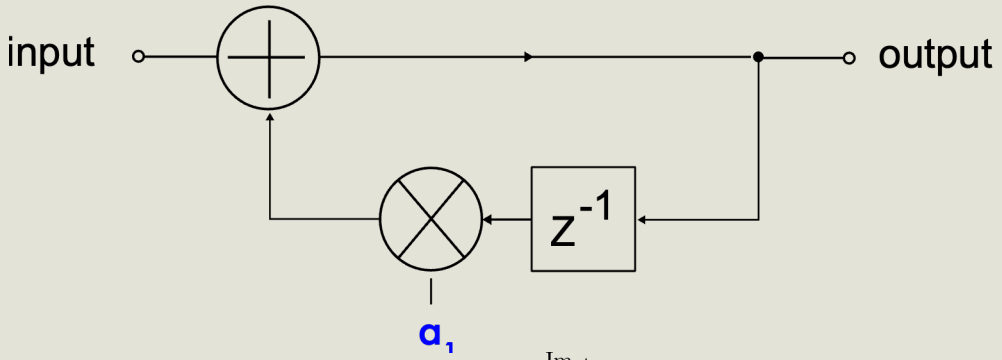
Difference Equation

$$y_n = x_n - a_1 y_{n-1}$$

Transfer Function

$$H(z) = \frac{1}{1 + a_1 z^{-1}}$$

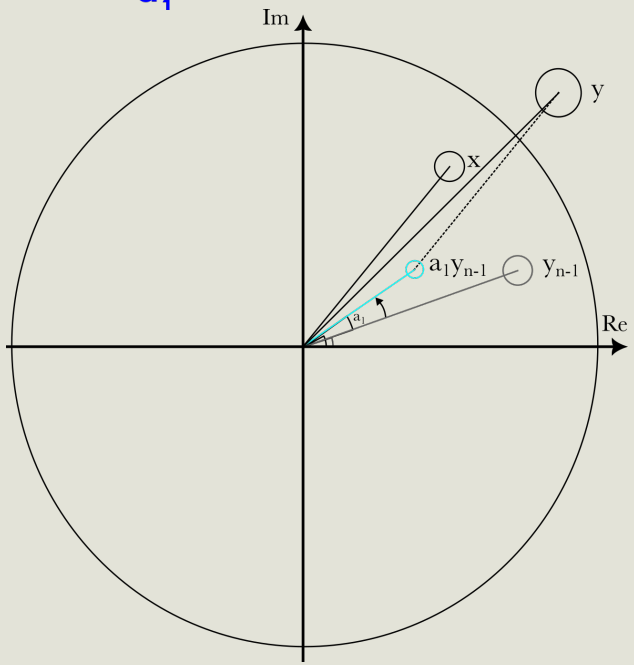
Finding a_1



$$H(z) = \frac{1}{1 + a_1 z^{-1}}$$

$$|H(z)| = \left|\frac{1}{1 + a_1 z^{-1}}\right| = \frac{1}{|z + a_1|}$$

$$|z + a_1| = 0 \longrightarrow z = -a_1$$



So to achieve a peak at a particular frequency ω we can choose z and set

$$a_1 = -z$$

$$z = e^{i\omega}$$

Focusrite novation

Back to the paper

Difference Equation

$$y_n = x_n - a_1 y_{n-1}$$

$$y_n = -a_1 y_{n-1}$$

$$y_n = a_1 y_{n-1}$$

Paper equation 1

$$z(n+1) = z_1 z(n), \quad (1)$$

$$z_1 \triangleq e^{j\theta_1} = \cos(\theta_1) + j\sin(\theta_1).$$

Implementing the paper filter

$$x(n+1) = x_1 x(n) - y_1 y(n) + u(n) \quad (12)$$
$$y(n+1) = y_1 x(n) + x_1 y(n) \quad (13)$$

$$y_n = a_1 y_{n-1}$$

$$y = \mathrm{r} + \mathrm{c}i \qquad a_1 = \mathrm{Re}\{a_1\} + \mathrm{Im}\{a_1\}i$$

$$(\mathrm{r_n} + \mathrm{c_n}i) = a_1(\mathrm{r_{n-1}} + \mathrm{c_{n-1}}i) \qquad y_n = (\mathrm{Re}\{a_1\} + \mathrm{Im}\{a_1\}i)(\mathrm{r_{n-1}} + \mathrm{c_{n-1}}i)$$

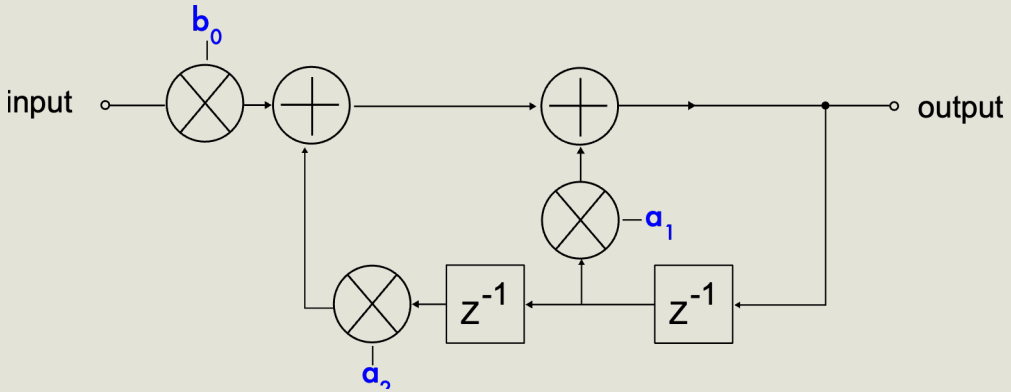
$$y_n = Re\{a_1\}\mathrm{r_{n-1}} + Re\{a_1\}\mathrm{c_{n-1}}i + Im\{a_1\}\mathrm{r_{n-1}}i + Im\{a_1\}\mathrm{c_{n-1}}i^2$$

$$= Re\{a_1\}\mathrm{r_{n-1}} - Im\{a_1\}\mathrm{c_{n-1}} + (Re\{a_1\}\mathrm{c_{n-1}} + Im\{a_1\}\mathrm{r_{n-1}})i$$

$$\mathrm{r}_n = Re\{a_1\}\mathrm{r_{n-1}} - Im\{a_1\}\mathrm{c_{n-1}} \qquad \mathrm{c}_n = Re\{a_1\}\mathrm{c_{n-1}} + Im\{a_1\}\mathrm{r_{n-1}}$$

Focusrite novation

Filters 9



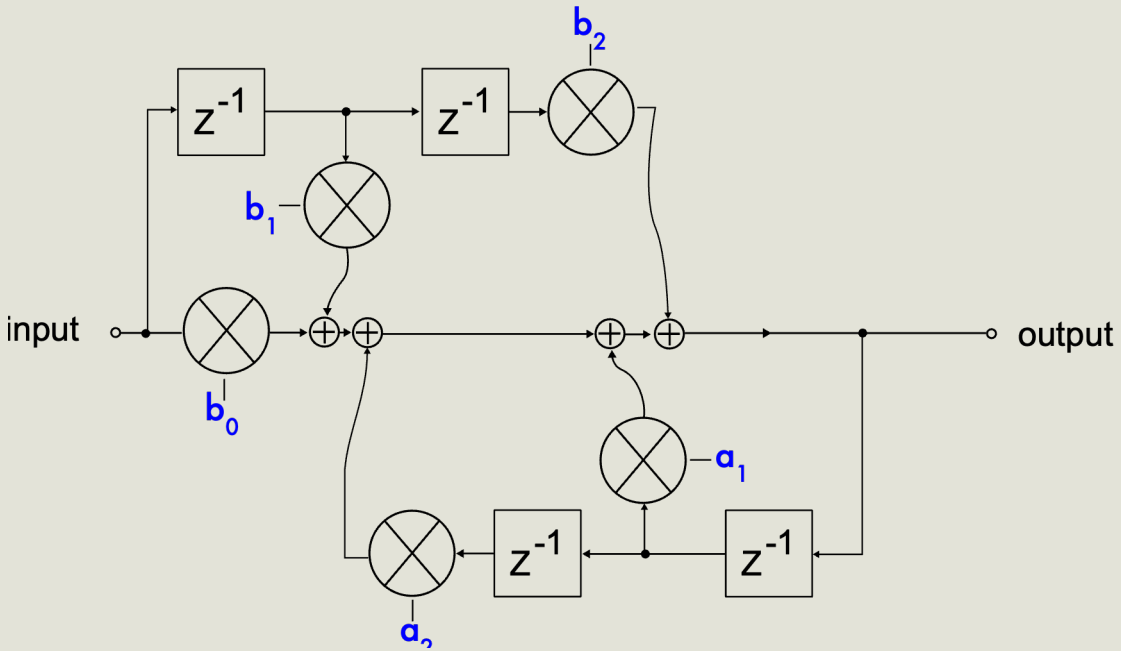
Difference Equation

$$y_n = b_0 x_n - a_1 y_{n-1} - a_2 y_{n-1}$$

Transfer Function

$$H(z) = \frac{b_0}{1 + a_1 z^{-1} + a_2 z^{-1}}$$

Filters 10



Difference Equation

$$y_n = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} - a_1 y_{n-1} - a_2 y_{n-1}$$

Transfer Function

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-1}}$$

The paper filter as a Biquad

Transfer Function

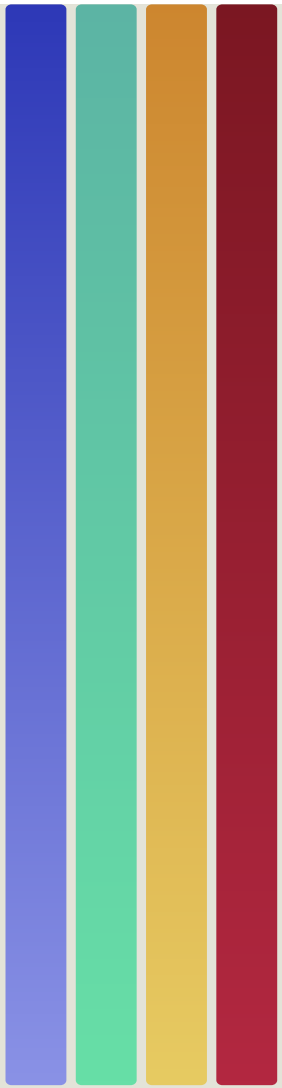
$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-1}}$$

$$\begin{aligned} H(z) &\triangleq \frac{Y(z)}{U(z)} = \frac{y_1 z^{-2}}{1 - 2x_1 z^{-1} + (x_1^2 + y_1^2) z^{-2}} \\ &= \frac{r_1 \sin(\theta_1) z^{-2}}{1 - 2r_1 \cos(\theta_1) z^{-1} + r_1^2 z^{-2}} \end{aligned}$$

$b_0 = 0$
$b_1 = 0$ $\quad\quad$ $\mathrm{a}_1 = -2\mathrm{r}_1 \cos(\theta_1)$
$b_2 = \mathrm{r}_1 \sin(\theta_1)$ $\quad\quad$ $b_2 = {\mathrm{r}_1}^2$

Focusrite novation

15 Minute Break



Focusrite novation

Credits

All material related to the plugin, testing and profiling and assistance with the Jupyter notebooks.

Edwin Tomboza

Encouragement and advice

Elvira Burdiel &Tim Lloyd @ Sonnox

Beta testing and feedback

Dan Mitchell

Emma Fitzmaurice

Pete Carss

James Hallowell

Tim Lloyd

Tom Damen-Lane

Presenting and facilitating today.

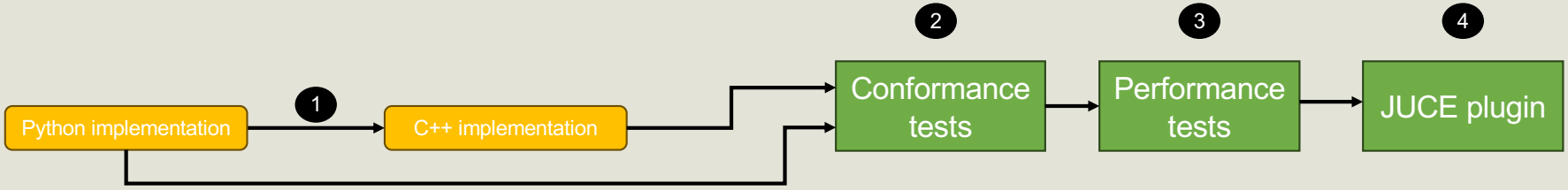
Ross Chisholm

Joel Ross

James Hallowell

Focusrite novation

From Python to C++ plugin



Focusrite novation

1. C++ implementation

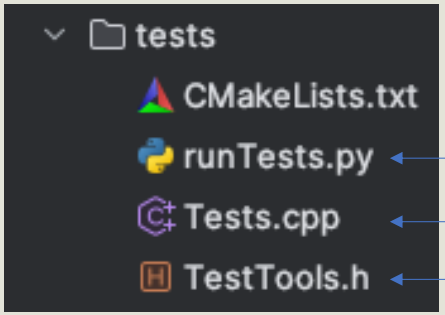
Task: Fill-in the functions in **source/MVMFilter.cpp**. The header is available and more description can be found below

```
class MVMFilter
{
public:

   // Used parameters to set the memory
   explicit MVMFilter(const Parameters& parameters);

   // Process single sample at a time
   double process(double);

   // Set the memory with the new frequency and new tau
   void set(double f, double tau);

private:
   // states
};
```

2. Conformance tests

Goal: Running the same test (similar input and similar parameters) with the python and C++ implementation and computing some metrics to verify that both output are matching.

Description:



Create the input signal, run the C++ tests containing the filter implementation in a subprocess, retrieve the result through the standard output stream, run the python implementation and compare both output

Define the C++ tests

C++ helper class to create the pipeline between C++ and python implementation

Expected output:

After running the test program, the terminal will print a PASS or FAIL message. A failed test is defined as getting an RMS error > -120dB which indicates an audible difference between both outputs.

A plot will be generated to visually see the test results.

$$\mathrm{RMSE}(y_{\mathrm{cpp}}, y_{\mathrm{python}}) = \frac{\sum_{i=0}^{N-1}(y_{\mathrm{cpp}} - y_{\mathrm{python}})^2}{N}$$

RMS error

Focusrite novation

3. Profiling

- Goal: Comparing processing usage of filter implementation from paper vs naïve implementation

- Description:
  - mvmFilterProcess process the input with the implementation from the paper
  - sineProcess process the input with the implementation using std::sine
  - We are using std::chrono::steady_clock::now() to start and stop timer before and after each function
  - Total time for each process is calculated and display in the terminal
  - Note: volatile keyword is used to tell the compiler to avoid any optimization which could impact the profiling. We are running each process NumberOfProcessingLoop=100 times to make it the process long enough and to get a better timer accuracy.

4. JUCE plugin – Setup plugin project with Projucer

0. Set global paths

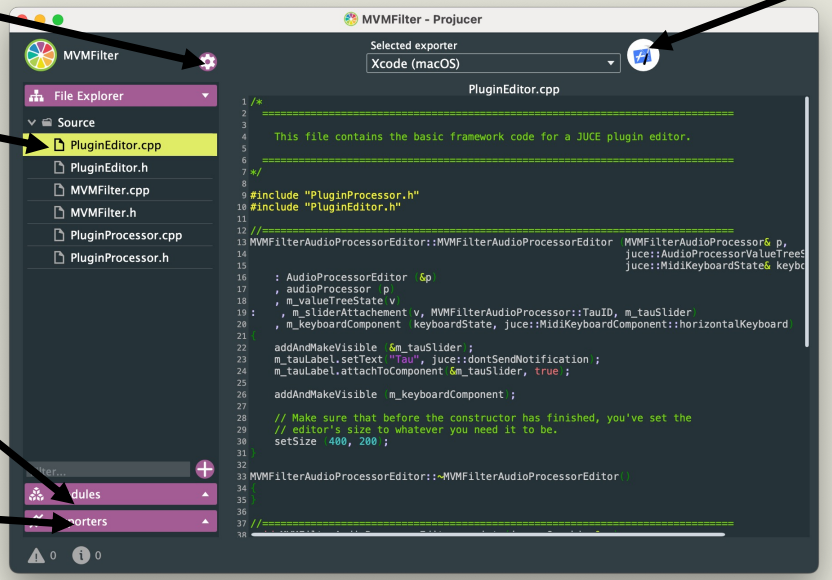
1. Set project configuration

2. Add source file

3. Add JUCE modules

4. Set build configuration

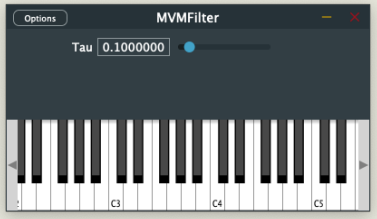
5. Export to IDE



Focusrite novation

4. JUCE plugin – Build and run plugin with IDE

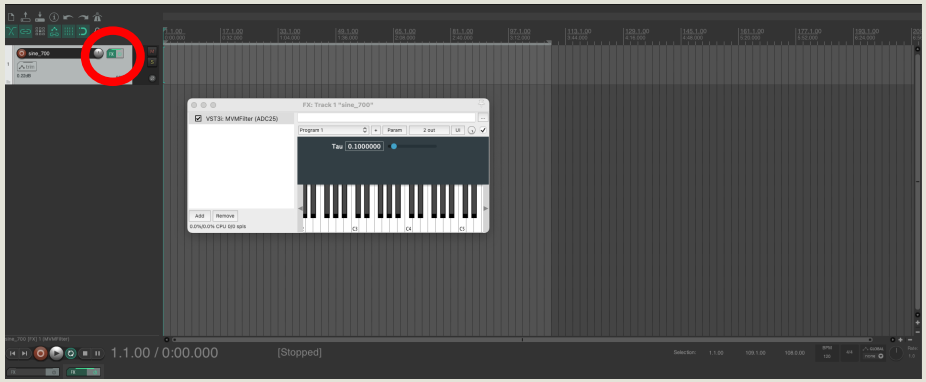
- MVMFilter – Standalone : Build standalone app and open it



- MVMFilter – VST3 and MVMFilter AU : Build VST3 plugin and copy them in the plugin folder (Windows)

Tau = 0.1

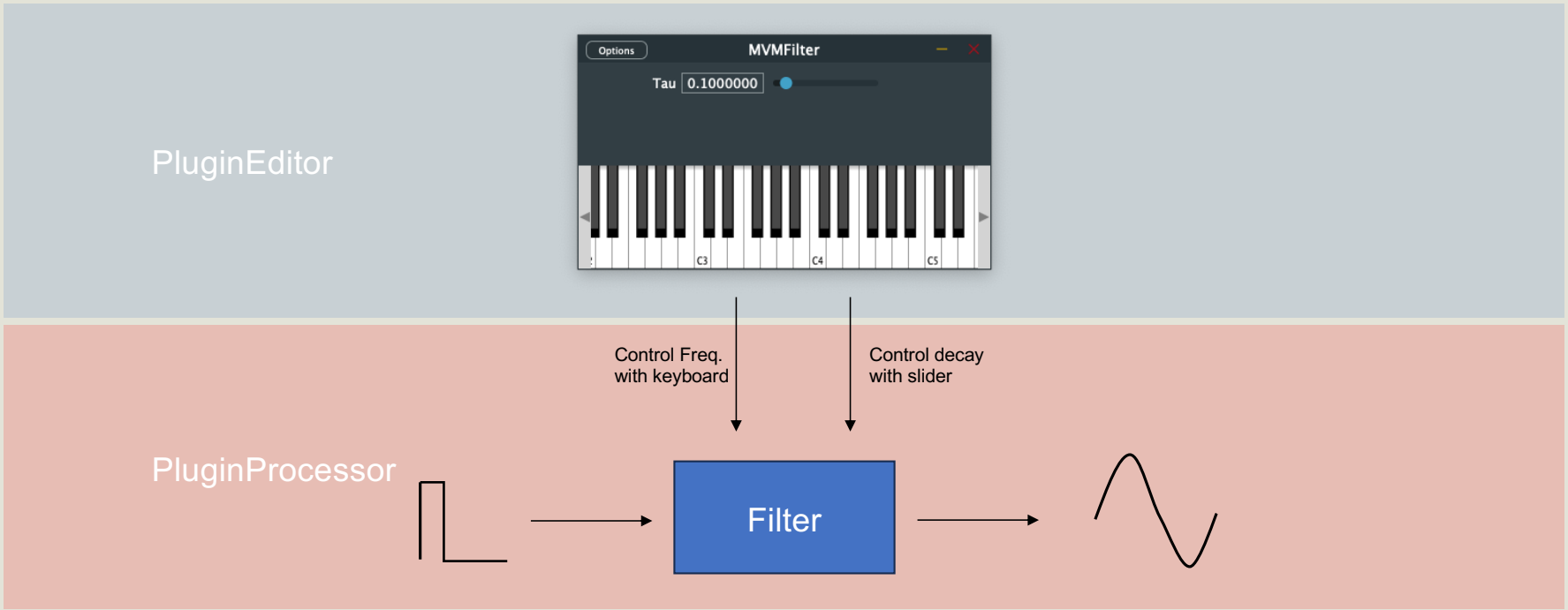
Tau = 0.3



Focusrite novation

4. JUCE plugin – Plugin description

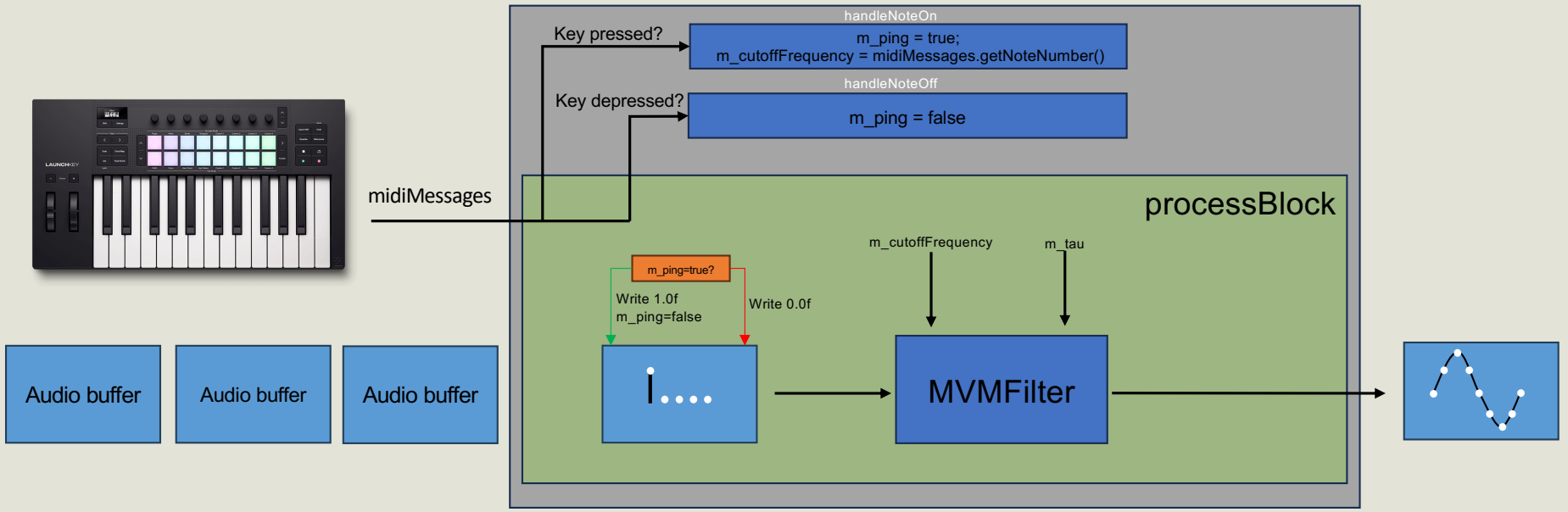
Using the filter and an audio source (impulse) to create a one-oscillator synth



Focusrite novation

4. JUCE plugin – Plugin description

PluginProcessor

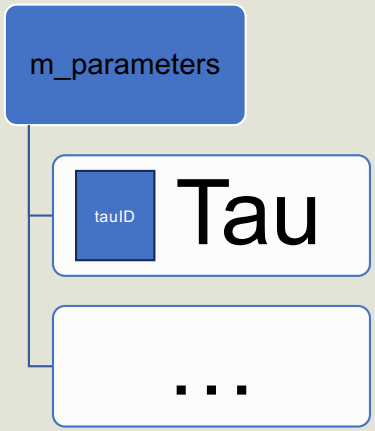


Focusrite novation

4. JUCE plugin – Code description

PluginProcessor

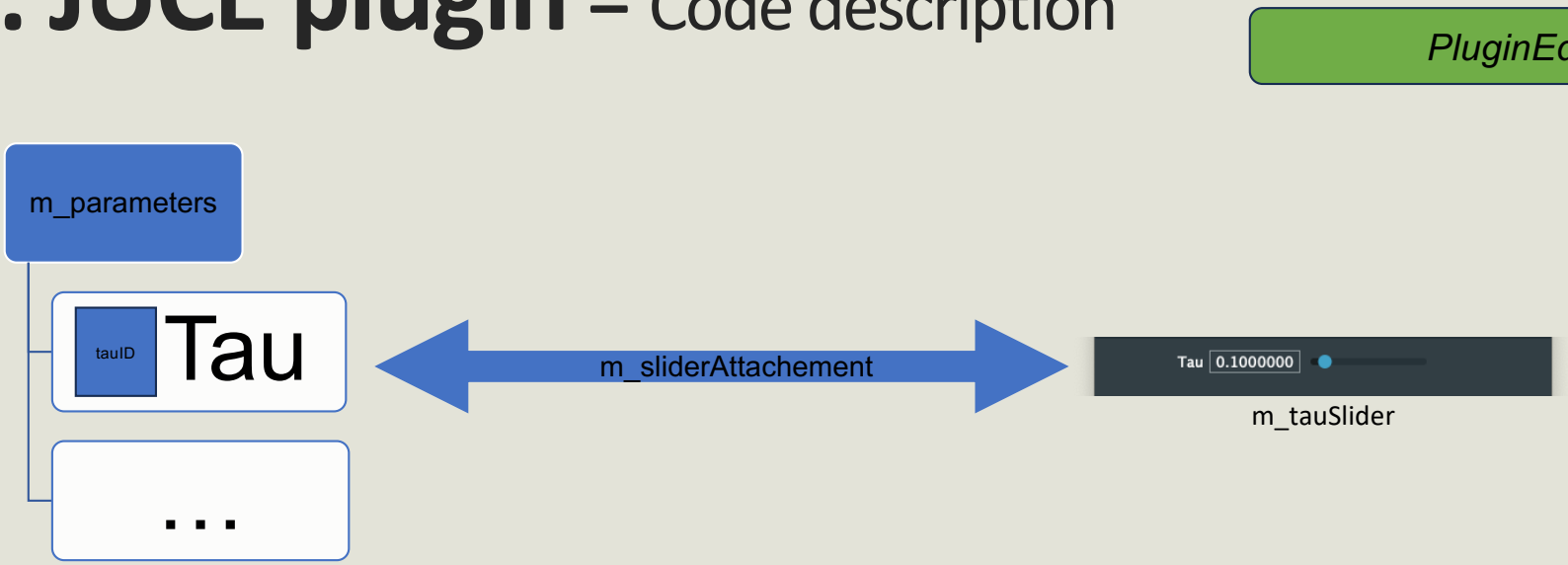
Parameters



- Tree storing parameters
- Parameter's value is retrieved with const char* IDs
- Shared between DSP (pluginProcessor) and UI (pluginEditor) for easy sync-up between both
- Extendable (need to be set in the pluginProcessor constructor)

Focusrite novation

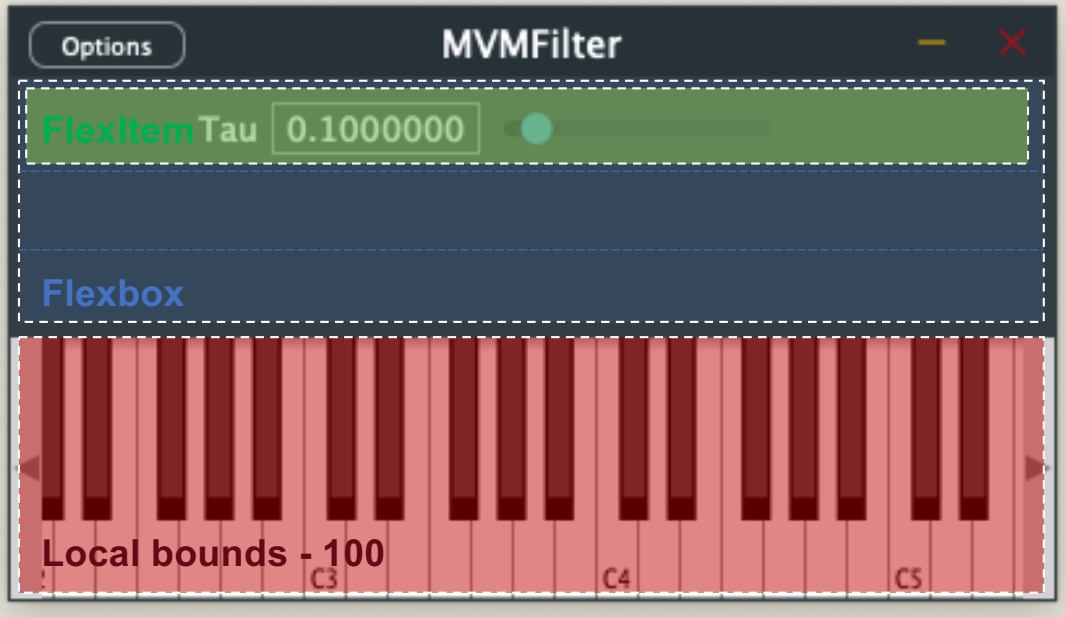
4. JUCE plugin – Code description



1. m_sliderAttachement(v, MVMFilterAudioProcessor::TauID, m_tauSlider) // Create bidirectional communication between slider on UI and backend parameters for DSP processing
2. addAndMakeVisible (&m_tauSlider) // Register the slider as a visual element to make it visible

4. JUCE plugin – Code description

PluginEditor:
MVMFilterAudioProcessorEditor::resized

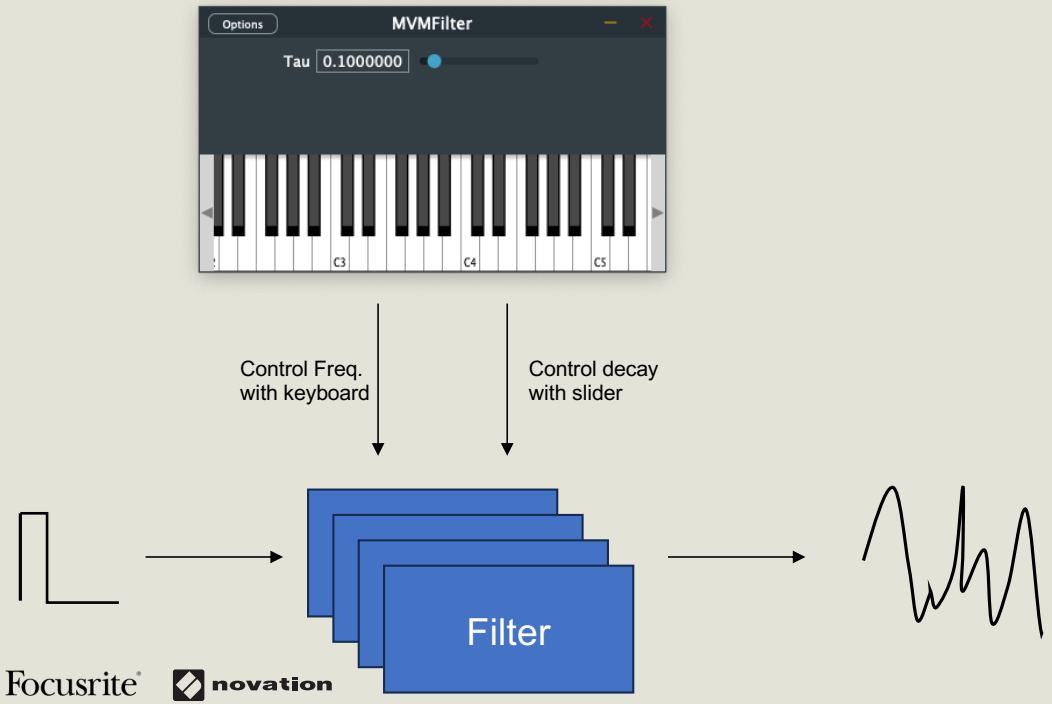


4. JUCE plugin – Advanced plugin 1

Tau = 0.01

Tau = 0.3

Adding more filter to create an oscillator bank:



Hints

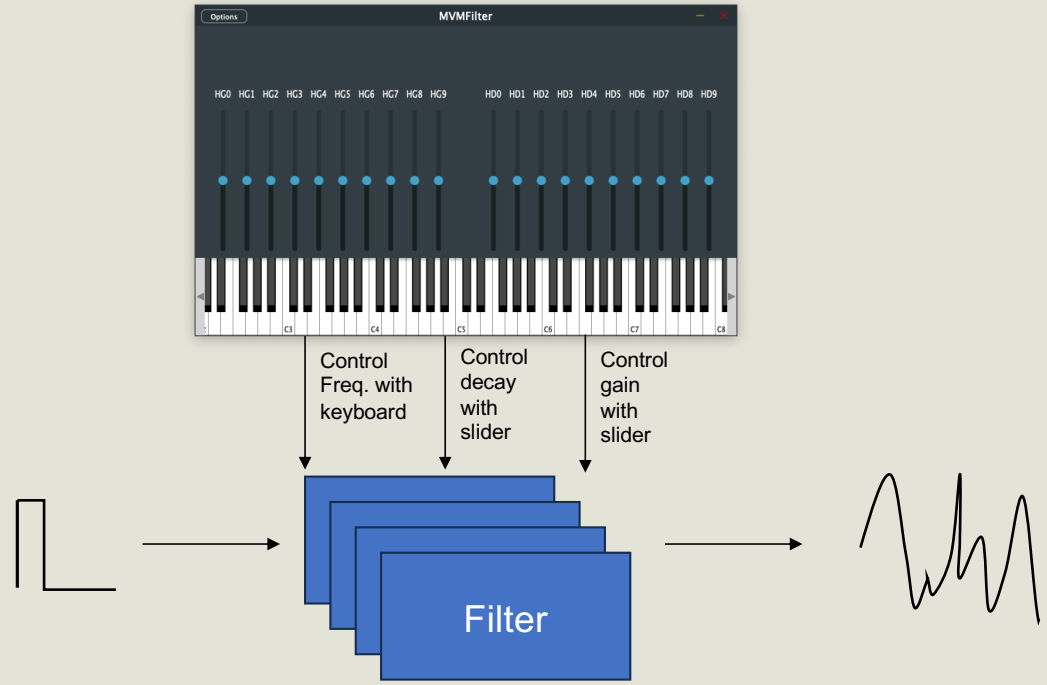
1. All modifications on the DSP side so PluginProcessor.h / PluginProcessor.cpp
2. Add more filters in the array
3. Iterate over all the filters in the processing function and add-accumulate their outputs when writing to the output buffer
4. Divide final sample by number of Filters to avoid saturation
5. Set the cutoff frequency and the tau differently for each filter to get more interesting sounding output

4. JUCE plugin – Advanced plugin 2

Tau = 0.5
gain = 0.5

Only H1, H8 and H9

Add UI to control the filters:



Hints

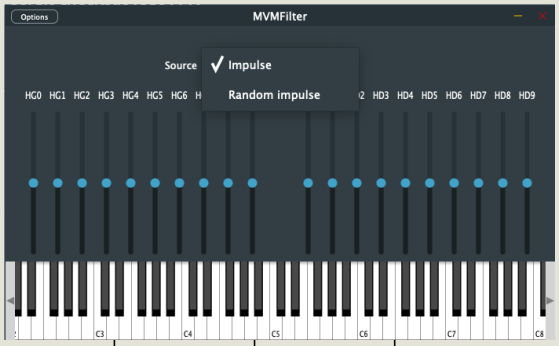
1. Add new parameters (gain and tau per filter) to parameter tree
2. Add new sliderAttachments in PluginEditor.h for each new parameter
3. Make each new slider visible
4. Add each new slider to the flexBox to arrange them in space

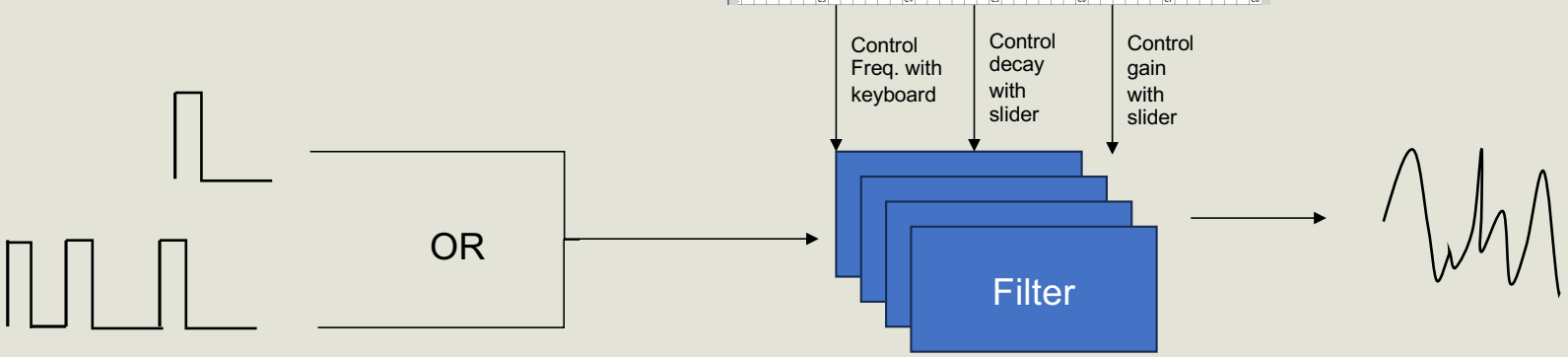
Focusrite novation

4. JUCE plugin – Advanced plugin 3

Add more input source:

Loud volume with random impulse





Focusrite novation

4. JUCE plugin – Advanced plugin 3

Hints DSP

1. Use juce::AudioParameterChoice for a parameters with options
2. Add new parameter to parameter tree
3. Use available random number generator from standard library

Hints UI

1. Use juce::ComboBox as a visual element to select between options
2. Use juce::ComboBoxAttachment to link the UI component and the backend parameter