Pure Data Module

Last Updated: May 6, 2011

Note:
This is the manual for the Pure Data (Pd) Module. Within this directory you will find the source for Pd and pd-l2ork (source and precompiled binary), several Pd tutorials and the patches that accompany them, as well as other pertinent information to the module.

Brief Introduction to Pure Data

Pure Data (or Pd) is a visual programming language developed by Miller Puckette in the 1990s for the creation of interactive computer music and multimedia works. Though Puckette is the primary author of the software, Pd is an open source project and has a large developer base working on new extensions to the program (http://en.wikipedia.org/wiki/Pure_Data).

Although the module focuses on the main development branch of Pure Data, this project was spawned out of a need for better documentation in the pd-l2ork branch of Pd. The Linux Laptop Orchestra (l2ork) at Virginia Tech uses this iteration of Pd to meet their needs for mixed media performances. More info can be found at http://code.google.com/p/pd-l2ork/ (code repository) and http://l2ork.music.vt.edu/ (L2ork website).

Motivation

The main motivation for this module is that Pure Data can often be a hard tool for new users to get familiar with. The GUI is not very polished and the website can often be hard to navigate. The need for an aggregation of good resources and clear tutorials is apparent for this sophisticated multimedia application.

Another motivation for this project is to promote better documentation standards in Pd. Under the direction of Ico Bukic of the Music Department at VT, I (Conor Scott) was asked to help re-document the entire cyclone library in Pd. Cyclone is a major compatibility library with Max/MSP, and many of the help files associated with objects in this library are cluttered and unclear. Jonathan Wilkes of VT created a help file standard, and this was used as a template for the new cyclone help files. This documentation, in particular, will help Max/MSP users transfer over to Pd more smoothly. Currently, this documentation is just merged in the pd-l2ork branch of Pd, but a long-term goal of this project is to have Pd mainline accept these changes as standard.

1. Module name

Pure Data (Pd) Module

2. Scope

This module is designed as an introductory tool to get one started with Pd. It covers downloading and installation instructions for multiple platforms, instructions on navigating the Pd GUI, and helps the user create simple patches to get started with Pd and generating electronic music. In addition, this

module contains many other resources (ebooks, tutorials, etc.) that give a very detailed account of Pd libraries and functionality, as well tutorials for advanced users.

3. Learning objectives

The user of this module should ultimately be able to understand the usage of all the different types of Pd messages, basic signal processing, and manipulation of the Pd window interface, as well as effective usage of help files. From this foundation, the user should feel comfortable with the basic features Pd has to offer and will have the foundation to learn more advanced topics through other resources in the module.

4. 5S characteristics of the module

a. Streams: this module deals with music data streams.

b. Structures: multiple objects are interconnected with each other and form a certain structure to manipulate sound files.

c. Spaces: users interact with Pd on its graphical user interface, which is a two dimensional space that is designed to help manipulation of sound files.

d. Scenarios: input sound files are manipulated by a series of objects applied in sequence. These series of steps are scenarios.

5. Level of effort required

a. Basic Tutorials < 2 hour (including configuration)

b. Advanced Tutorials via E-book, ~10 hours per book

6. Relationships with other modules

Module 2-b: Multimedia can be taught prior to this Pd module. However, Module 2-b has not been developed yet.

7. Prerequisite knowledge

The user of this module should does not need much programming knowledge, and does not have to be musically inclined. It would, however, be very helpful for the user to have a basic understanding of digital signal processing (types of waves and filters), musical harmonics, and advanced algebra/geometry to simulate different types of musical phenomena. Without this knowledge the tutorials are very doable, but making electronic music beyond the tutorials may lead to unexpected results and horrible sounds!

8. Introductory remedial instruction

None.

9. Body of Knowledge

A. More Information on Pd

Pure Data (or Pd) is a real-time graphical programming environment for audio, video, and graphical processing. Pure Data is commonly used for live music performance, VeeJaying, sound effects, composition, audio analysis, interfacing with sensors, using cameras, controlling robots or even interacting with websites. Because all of these various media are handled as digital data within the program, many fascinating opportunities for cross-synthesis between them exist. Sound can be used to manipulate video, which could then be streamed over the internet to another computer which might analyze that video and use it to control a motor-driven installation.

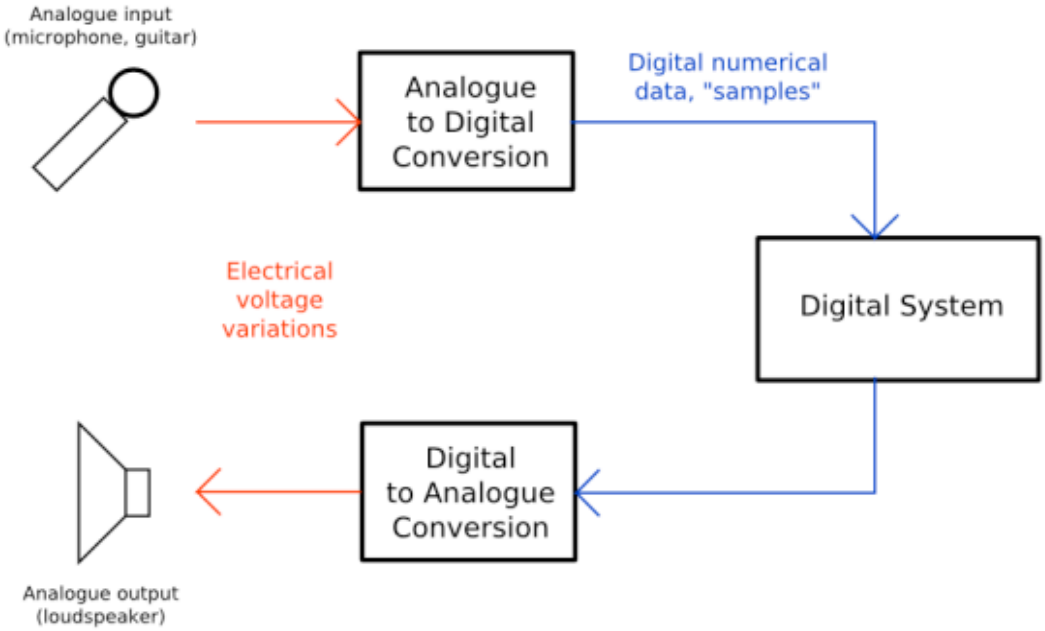
Programming with Pure Data is a unique interaction that is much closer to the experience of manipulating things in the physical world. The most basic unit of functionality is a box, and the program is formed by connecting these boxes together into diagrams that both represent the flow of data while actually performing the operations mapped out in the diagram. The program itself is always running, there is no separation between writing the program and running the program, and each action takes effect the moment it is completed.

The community of users and programmers around Pure Data have created additional functions (called "externals" or "external libraries") which are used for a wide variety of other purposes, such as video processing, the playback and streaming of MP3s or Quicktime video, the manipulation and display of 3-dimensional objects and the modeling of virtual physical objects. There is a wide range of external libraries available which give Pure Data additional features. Just about any kind of programming is feasible using Pure Data as long as there are externals libraries which provide the most basic units of functionality required.

The core of Pure Data written and maintained by Miller S. Puckette (http://crca.ucsd.edu/~msp/) and includes the work of many developers (http://www.puredata.org/), making the whole package very much a community effort. Pd runs on GNU/Linux, Windows, and Mac OS X, as well as mobile platforms like Maemo, iPhoneOS, and Android (Efloss Manual).

B. Background on Digital Music

An important feature to notice when working with music on a computer is that often the data is required to transfer back and forth between analogue and digital format. The graphic below illustrates how analogue data is input and converted to digital data using an Analogue to Digital Converter (adc), manipulated in some way digitally, and then converted back to analogue using a Digital to Analogue converter (dac). This point will be made clearer throughout the tutorials.



Pd is special tool in that it does “real-time” audio processing. This means that changes made to your Pd program can be noticed, and often heard, immediately. This can be very important for people who manipulate digital music in a live performance setting.

C. Installation

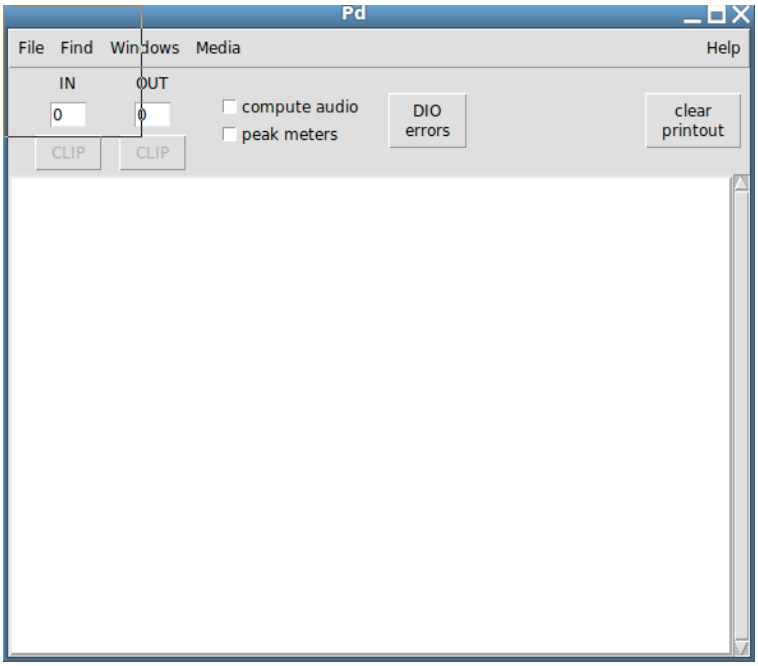
Pure data can be found and downloaded at www.puredata.info. Versions available are For Mac, Windows, and all major distros of Linux. Both source and precompiled binaries are available. In Linux, Pd can also be installed through most package managers (apt, yum, etc.).

The pd-l2ork version can also be downloaded and installed from http://l2ork.music.vt.edu/. This version is only supported on Debian system (Debian, Ubuntu, Linux Mint, other Debian derivatives...), but contains all the features of regular Pd and more, such as Wiimote drivers, additional network support, and better documentation (especially in cyclone library!).

More detailed information on installation can be found in the Efloss manual inside this module in the /Efloss Manual directory.

D. Configuration

After installing Pd, the first thing to do is make sure audio is configured properly. After starting Pd, one should arrive at the main window that looks something like this (varies per version):

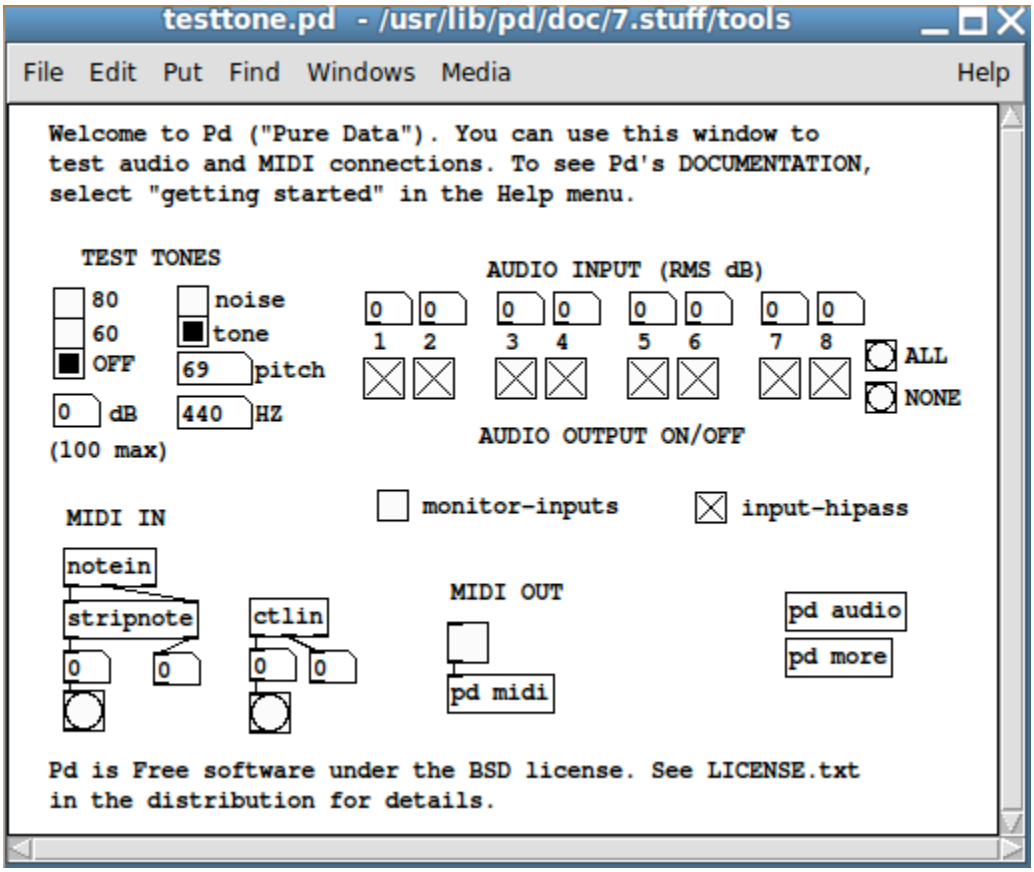


In order to test audio, you must click the “computer audio” toggle and go to the Media menu and select one of the following:

OSX : Media -> portaudio/jack
Linux : Media -> OSS/ALSA/jack
Windows : Media -> ASIO (via portaudio)

These programs will process your sound in real time. While either of these may work, jack is recommended for Ubuntu and OSX (jack must be manually installed).

After selecting the audio driver, click back on the Media menu and select “Test Audio and MIDI”. A Pd patch file will pop up that looks something like this:



Under Test Tones, change to 60 or 80. If your audio is set up correctly, you should hear a noise. You can also mess with pitch and HZ value to manipulate what you are hearing.

If you do not hear any music you may not have audio set up correctly, or your audio daemon may be busy with other processing. Make sure you have closed all other applications that could be playing sound (iTunes, Media Player, etc.) and try again. If the problem persists, check the Efloss Manual for more detailed audio setup instructions and troubleshooting.

E. Notes on Interface

While the main functionality of Pure Data doesn't change between operating systems, the locations and contents of some of the menus do. This tutorial will focus on the Linux version of Pd, but on page 55 of the Efloss Manual you will find instructions for other versions of Pd.

On Linux you will be able to do the following:

From the "File" menu, you can:

1. Create a "New" Pd patch
2. "Open" a Pd patch which is saved on your computer
3. Send a "Message" to the running Pd application
4. Set the search "Path" which Pd uses
5. Change the "Startup" flags which Pd uses
6. "Quit" Pd

From the "Find" menu, you can:

1 ."Find last error" which occurred in the program

From the "Windows" menu, you can:

1. Change between the different open Pd patches

From the "Media" menu, you can:

1. Turn audio "ON" and "OFF"
2. Change between the different available audio drivers
3. Change between the different available MIDI drivers
4. Change the "Audio Settings"
5. Change the "MIDI Settings"
6. "Test Audio and MIDI"
7. View the CPU "Load Meter"

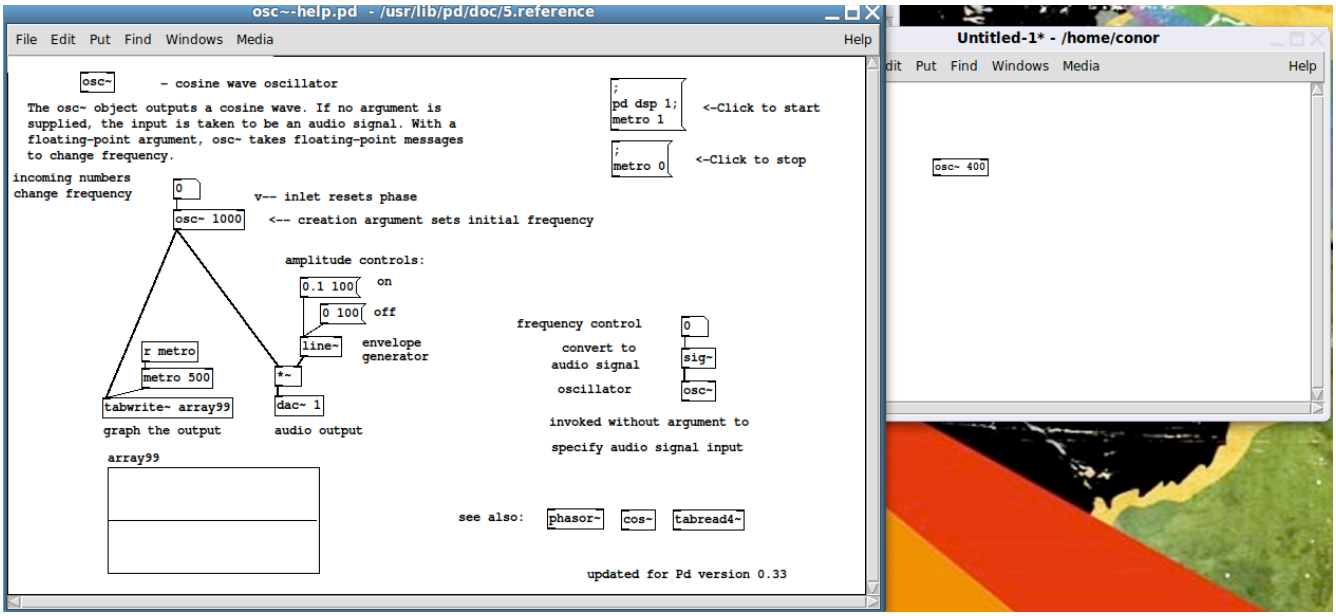
F. Interacting with GUI

Use the "Put" menu to place an "Object" in your patch. Click on the patch to drop the object in its place. You will see a box made of a broken blue line, with a flashing cursor inside indicating that you should type something there.

Objects are the "vocabulary" of Pd. The more names of objects you know, the more complicated things you can do with Pd. If you type the word "print" inside this object and click again outside the box, you will create the [print] object.

Return to the "Put" menu, and this time place a "Number" in your patch. Notice that the shape of the number box is different from the shape of the object box.

To get help associated with a specific object you can right-click it, then just select "help" from the drop-down menu (pictured below). This is something like the "dictionary entry" for the object, and should define what it does and also show several examples of its use.

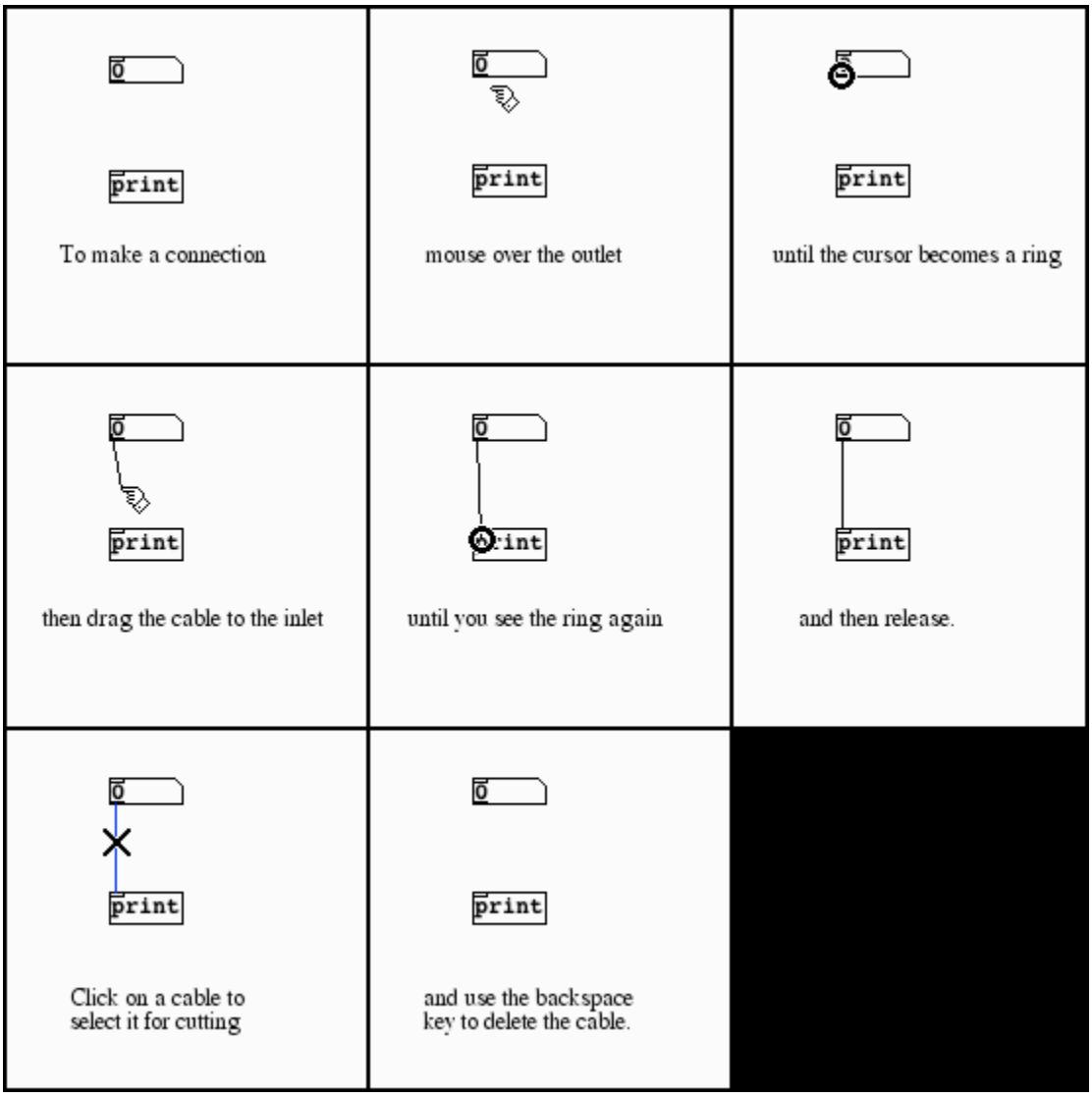


* Figure shows clicking on help button for osc~ object, brings up help file for object osc~

A nice thing to note about Pd help files are patches themselves and the examples presented in them can be highlighted and directly copy and pasted into other patch windows.

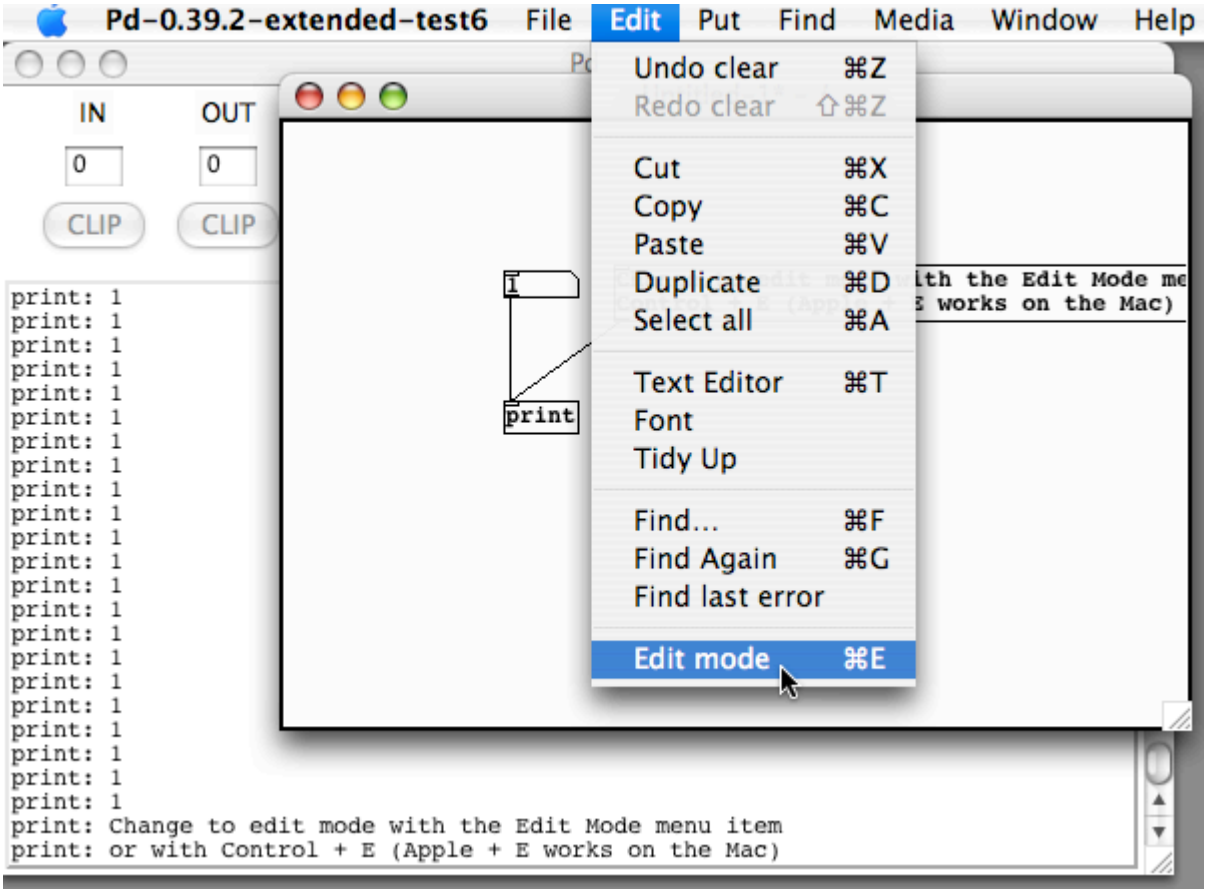
You should also notice that both the object and the number boxes have small rectangles at the corners. If these are at the top of the object, they are called "inlets", and at the bottom they are called "outlets". When you are working on your patch, your cursor is shaped like a pointing finger. If you put that finger over an outlet, it changes into a black circle which indicates that the outlet is selected.

Select the outlet of the number box, click and drag that black circle until it reaches the inlet at the top of the [print] object. When you have done that, you will see the cursor change from the pointing finger to the black circle again. If let go of the mouse button now, you will make a connection from the outlet of the number box to the inlet of [print]. If you want to remove this connection, place your cursor over the connection until you see a black X and then click. The connection will turn blue and you can remove it with the Backspace or Delete key on your keyboard. Below is a figure showing this:



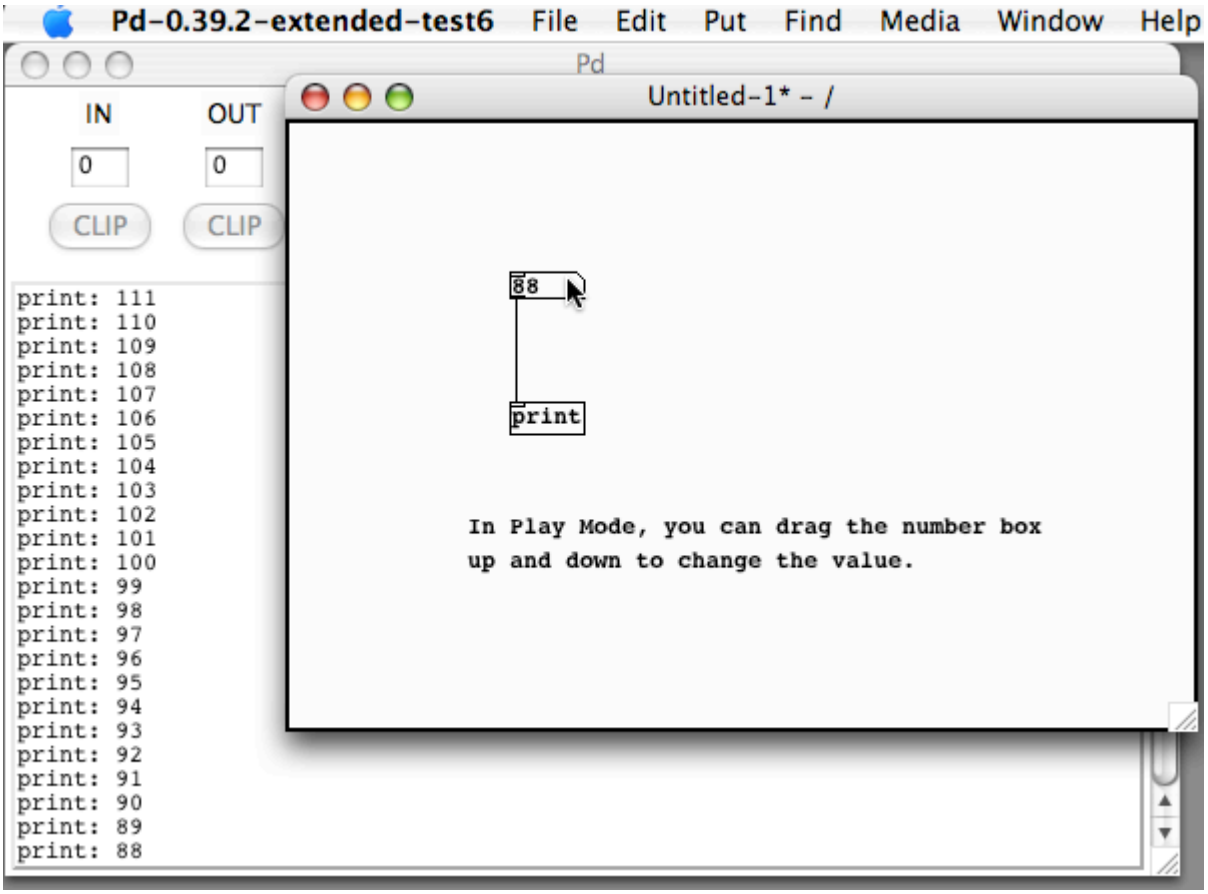
While in edit mode you can drag the mouse over the areas objects occupy and do operations such as move, copy, cut, and paste.

So far we've been able to put objects in the patch, connect them, move them around or delete them. But how does one get some results from this patch? In this case, we have connected a number box to a [print] object, which should print the numbers we send to it in the main Pd window. Now we have to learn how to send numbers to the print object. To make this happen, we need to change out of "Edit Mode" and into "Play Mode". You can do this by clicking on the "Edit Mode" item in the Edit menu, or by using the Control and E keys . The figure below shows this in the GUI:



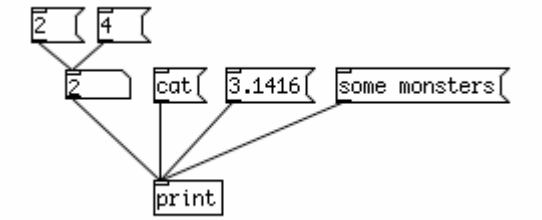
When you enter into "Play Mode", you will see that the pointing finger cursor changes into an arrow cursor. This change signifies that the mouse will interact differently with the elements within the patch.

Now, if you click inside the Number object and drag your mouse up or down you can change the numbers inside of it. Any changed number is sent out through the outlet to any connected objects. Since the object is connected to the inlet of the [print] object the number is printed to the main Pd window. The figure below shows this:



Messages, Symbols, Comments

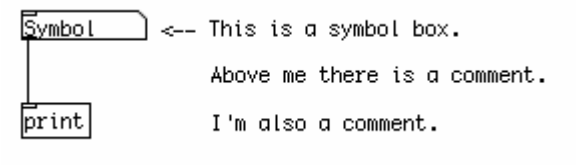
The "**Message**" box is used to store and send information to other objects, and can contain numbers or text. It also has a unique shape, which resembles an envelope like you would use to send a letter. Place two different messages above the number box in our exercise. Like the object, messages also give a flashing cursor indicating that you should enter some information when you create them. Enter "2" in one of the messages and "4" in the other, and connect both to your number box. Switch to Play Mode and click on each of the messages. When you do, you will see that the number box changes according to the message that you send it, and that the message is also sent onwards to the [print] object.



A "**symbo**l" is another way of storing and sending information. Once created, you can use it to display the output of some objects, or you can type directly into it and hit Enter to send the text out. Please note that no spaces will appear in the symbol box when you type into it, since separate words would be considered separate symbols.

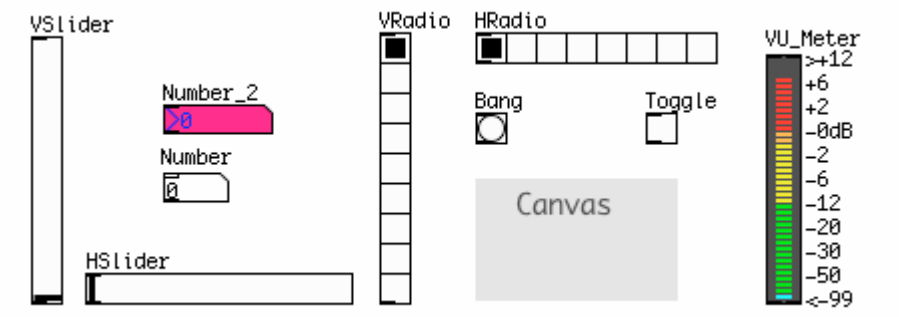
A "**comment"** is simply a way of making a note to yourself so that you (or someone else) can understand what you were trying to do later on. You can make as few or as many as you want, and they have no effect on the patch itself.

The figure below represents these two definitions more clearly:



GUI Objects

Pd has a number of GUI objects you can use to graphically control your patch and to improve its visual appearance. These are:



1. **Bang**: this GUI object sends a Message named "Bang" every time it is clicked. "Bang" is a special message, which many Objects interpret as "do an action right now!". Using the Bang GUI object is the same as creating a Message box with the word Bang in it. The Bang GUI object can also be used to receive and display Bang messages

2. **Toggle**: when clicked, the Toggle sends out one of two values--a zero when it is unchecked and a non-zero number when it is checked. The non-zero number is 1 by default; however this can be changed in the "Properties". The Toggle also has an inlet, which can be used to display whether an incoming number is zero or not.

3. **Number2**: this is almost identical to the Number box, however it has further options in its "Properties", including the ability to save its current value when the patch is saved (by changing the "no init" box to "init"). The Number2 has an inlet which can be used to display incoming numbers as well.

4. **Vslider and Hslider**: these are Vertical and Horizontal sliders which send out their current value when moved with the mouse. The default range of a slider is 0-127, which can be changed in the "Properties". Both sliders have an inlet which can be used to display incoming numbers within the range of the slider.

5. **Vradio and Hradio**: these are Vertical and Horizonal "radio buttons", which send out their current value when one of the buttons in them is clicked with the mouse. The default size of a radio button is 8 buttons, which can be changed in the "Properties". Both types of radio buttons have an inlet each, which can be used to display integer (whole) numbers within the range of the radio buttons.

6. **VU**: a VU meter displays the average volume level of any audio signal which is connected to it in Decibels. You may switch the value scale on the right side on and off in the "Properties".

7. **Canvas**: a canvas is a rectangular area of pixels, whose size and color may be changed under its "Properties". Canvases are useful as backgrounds in your patch to improve its visual appearance and readability. Canvas also can be used as movable GUI objects that gather information about their position (x,y) inside a patcher. Keep in mind that Pd remembers the order in which anything is placed in the patch, so if you want your canvas to be behind certain objects, you must either create it first, or you must Select, Cut and Paste the objects you want in the foreground so that they appear in front of the canvas.

10. Resources

Within this directory are several folders:

I. /Efoss Manual for PD: pdf of PD tutorial made by EFOSS website
II. /Making Electronic Music in PD: Link to e-book and associated patch files for tutorials
III. /Theory and Techniques of Electronic Music: PDF made by creator of Pd, as well as associated patch files for tutorial
IV. /PD install archives: Tars for various versions of PD (explained below)
V. /Other Examples: dub-step generator and bass synthesizer example
VI. ./new_cyclone.zip: zip of new cyclone help files for pd-l2ork

From here, the user should have a very good idea of basic features available in Pd, as well as a good handle on working with the GUI. Knowing the basic modulation and filter techniques presented in the learning exercises is essential when making digital music. The following ebooks and tutorials included in the module should be sufficient to lead the user into advanced topics such as sampling, wave Shaping, modulation synthesis, granular synthesis, Fourier analysis, sequencing, and other topics that utilize the full power of Pd libraries.

1. Theory and Techniques of Electronic Music, by Miller S. Puckette (creator of PD)

This tutorial is contained in the module and also available at http://crca.ucsd.edu/~msp/techniques/latest/book-html/. This book covers many advanced topics in music and is not for those wanting a light introduction to Pd. Here you will find mathematical formulas supporting all the exciting operations you can do with Pd and their connection with musical phenomena. I would recommend this book for someone looking to have a very clear understanding of the mathematics behind digital music.

2. Making Electronic Music In PD, by Johannes Kreidler

This tutorial can be found at http://www.pd-tutorial.com/english/index.html and its associated patches are contained in the module. It assumes you have Pd installed and are somewhat familiar with the GUI, but it has very slow and easy step by step tutorials leading into advanced topics. This is very good tutorial for someone that already has a basic grasp on Pd and wants to start quickly playing with cool programs.

3. Efloss Manual of PD

This e-book is contained in the module(PDF) and can also be found at http://en.flossmanuals.net/pure-data/ch018_oscillators. This book contains step by step instructions for downloading and installing Pd, as well as baby steps to get started. This is probably the best tutorial for beginners, but quickly leads into advanced topics after the first few chapters. I would recommend this tutorial for a beginner who has little to no experience in programming or digital music.

4. Pure Data Website

http://www.puredata.info is the official website for Pure Data, and you can find resources on all things Pd related here. There are user forums, places to submit patches, documentation, and development resources. I find, however, that the website is a bit cluttered and at times hard to navigate. I would recommend crawling this site after understanding the basics of Pd, but would not direct a brand new user to this site to learn from scratch.

11. Concept map

None.

12. Exercises / Learning activities

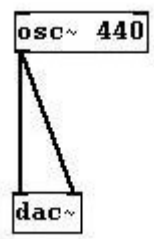
If all the previous information has been read, the user should feel comfortable enough with Pd to start doing some exercises that show basic features of Pd.

Exercise 1: Working with Oscillators

Oscillators are the basic signal generators in electronic music. By combining, filtering or modulating them, almost any imaginable sound can be created. In Pure Data, a stream of numbers between the values of -1 and 1 represents audio signals. So the waveform of each oscillator has been programmed to send out values within this range. The name of each oscillator refers to its waveform, which is the shape of one period (or one Hertz) of that oscillator. Different waveforms make different sounds.

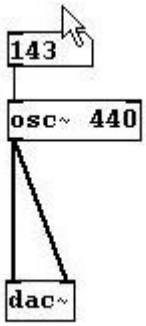
The Sine/Cosine Wave Oscillator makes a pure tone with no harmonics. The shape of the wave smoothly moves from 0 up to 1, back down through 0 to -1 and back up to 0. (Remember to turn the DSP on so that you can hear the audio).

Start by making an osc~ object connected to a dac~ (digital to audio converter):



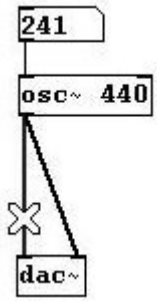
You should be hearing a sound from both speakers, specifically at 440 Hz (an A note). You should hear sound from both of the computer's speakers. Now create a number box (**Put Number** or with key command **Ctrl-3**) and attach its outlet to the inlet on the object named "osc~". Then you need to change

into what is called "Execute mode" (**Edit mode**, or with key command **Ctrl-E**; the cursor turns into an arrow). Click on the number box, hold the mouse button, and move the mouse up and down:

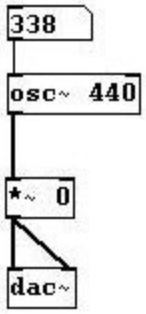


This changes the numbers and the pitch. The value should be at least 100; this range can be more finely adjusted by holding SHIFT while clicking and moving the mouse as described above. Another way to enter values into the number box is to click on the number box, enter a value on the keyboard, and press ENTER.

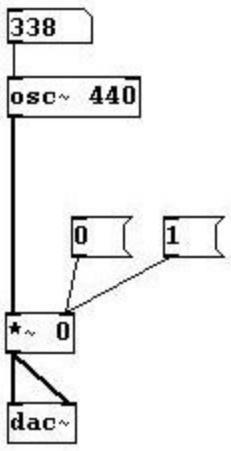
Now change back to the other mode, the "Edit mode" (**Edit → Edit mode**, or key command **Ctrl-E**). Move the cursor, which should have changed back to a hand, over the connection between "osc~" and "dac~". The cursor becomes an X. Click on it, which will turn the cable blue.



Then go to **Edit Cut** or simply press BACKSPACE. This terminates the connection. Cut the other connection to "dac~" as well. Now create a new object where the cables used to be: "*~ 0" and connect it to the other objects as shown:



Then create two "Message" boxes (**Put Message** or **Ctrl-2**) as below and enter "0" in one and "1" in the other.



Change back to execute mode (**Edit → Edit mode** or **Ctrl-E**) and click on the two message boxes in turn: clicking 1 turns the sound on, clicking 0 turns it off.

- In all the examples so far, notice the difference between the cable for messages, which is thin, and the cable for audio, which is thicker. Messages can be sent to audio objects (those with a ~ in their name), but usually audio cannot be sent to message objects (those without a ~ in their name). Attempting to do so will cause Pd to print "error: can't connect signal outlet to control inlet", and it will not allow the connection to be made.

Exercise 2: Working with MIDI

For many musical applications, the MIDI scale is a useful way of controlling the frequency of an oscillator. One can imagine the MIDI scale as a piano keyboard with 128 keys on it, and each key has been marked with a frequency in Hertz, which represents that musical note. Below is a part of the table, which makes up the MIDI scale. The most important thing to notice is that a note which is one octave higher than another note (for example, the three A notes of 110 Hz, 220 Hz and 440 Hz) has a frequency which is twice that of the lower note. The table below shows this:

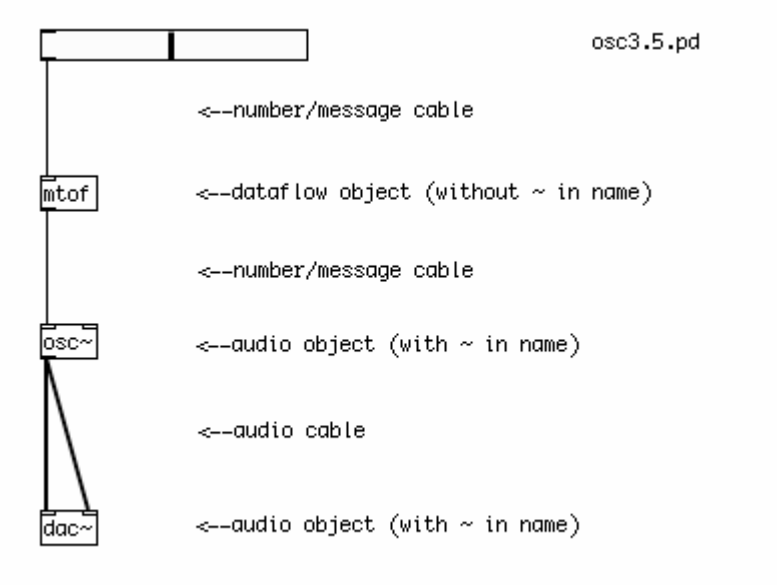
MIDI note to frequency conversion chart

MIDI Note	Frequency	MIDI Note	Frequency	MIDI Note	Frequency
Octave -1		**Octave 0**		**Octave 1**	
C 0	8.1757989156	12	16.3515978313	24	32.7031956626
Db 1	8.6619572180	13	17.3239144361	25	34.6478288721
D 2	9.1770239974	14	18.3540479948	26	36.7080959897
Eb 3	9.7227182413	15	19.4454364826	27	38.8908729653
E 4	10.3008611535	16	20.6017223071	28	41.2034446141
F 5	10.9133822323	17	21.8267644646	29	43.6535289291
Gb 6	11.5623257097	18	23.1246514195	30	46.2493028390
G 7	12.2498573744	19	24.4997147489	31	48.9994294977
Ab 8	12.9782717994	20	25.9565435987	32	51.9130871975
A 9	13.7500000000	21	27.5000000000	33	55.0000000000
Bb 10	14.5676175474	22	29.1352350949	34	58.2704701898
B 11	15.4338531643	23	30.8677063285	35	61.7354126570
Octave 2		**Octave 3**		**Octave 4**	
C 36	65.4063913251	48	130.8127826503	60	261.6255653006
Db 37	69.2956577442	49	138.5913154884	61	277.1826309769
D 38	73.4161919794	50	146.8323839587	62	293.6647679174
Eb 39	77.7817459305	51	155.5634918610	63	311.1269837221
E 40	82.4068892282	52	164.8137784564	64	329.6275569129
F 41	87.3070578583	53	174.6141157165	65	349.2282314330
Gb 42	92.4986056779	54	184.9972113558	66	369.9944227116
G 43	97.9988589954	55	195.9977179909	67	391.9954359817
Ab 44	103.8261743950	56	207.6523487900	68	415.3046975799
A 45	110.0000000000	57	220.0000000000	69	440.0000000000
Bb 46	116.5409403795	58	233.0818807590	70	466.1637615181
B 47	123.4708253140	59	246.9416506281	71	493.8833012561
Octave 5		**Octave 6**		**Octave 7**	
C 72	523.2511306012	84	1046.5022612024	96	2093.0045224048
Db 73	554.3652619537	85	1108.7305239075	97	2217.4610478150
D 74	587.3295358348	86	1174.6590716696	98	2349.3181433393
Eb 75	622.2539674442	87	1244.5079348883	99	2489.0158697766
E 76	659.2551138257	88	1318.5102276515	100	2637.0204553030
F 77	698.4564628660	89	1396.9129257320	101	2793.8258514640
Gb 78	739.9888454233	90	1479.9776908465	102	2959.9553816931
G 79	783.9908719635	91	1567.9817439270	103	3135.9634878540
Ab 80	830.6093951599	92	1661.2187903198	104	3322.4375806396
A 81	880.0000000000	93	1760.0000000000	105	3520.0000000000
Bb 82	932.3275230362	94	1864.6550460724	106	3729.3100921447
B 83	987.7666025122	95	1975.5332050245	107	3951.0664100490
Octave 8		**Octave 9**			
C 108	4186.0090448096	120	8372.0180896192		
Db 109	4434.9220956300	121	8869.8441912599		
D 110	4698.6362866785	122	9397.2725733570		
Eb 111	4978.0317395533	123	9956.0634791066		
E 112	5274.0409106059	124	10548.0818212118		
F 113	5587.6517029281	125	11175.3034058561		
Gb 114	5919.9107633862	126	11839.8215267723		
G 115	6271.9269757080	127	12543.8539514160		
Ab 116	6644.8751612791				
A 117	7040.0000000000				
Bb 118	7458.6201842894				
B 119	7902.1328200980				

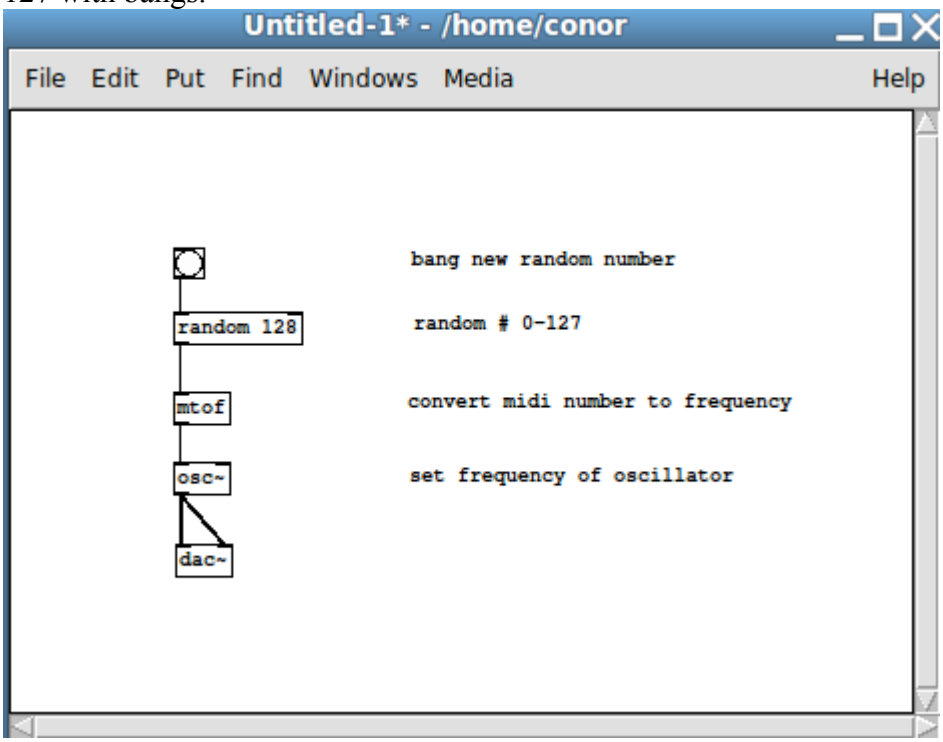
The object in Pd which turns a MIDI note into a frequency in Hertz is called [mtof], or MIDI to Frequency. When the MIDI note "69" is sent to it, for example, it will output the number "440".

Looking at our examples, you can see that each slider has a range of 0-127, and this is converted by an [mtof] object to a frequency, which tells the oscillator what to do.

The patch below uses a horizontal slider to convert midi numbers to these frequencies:

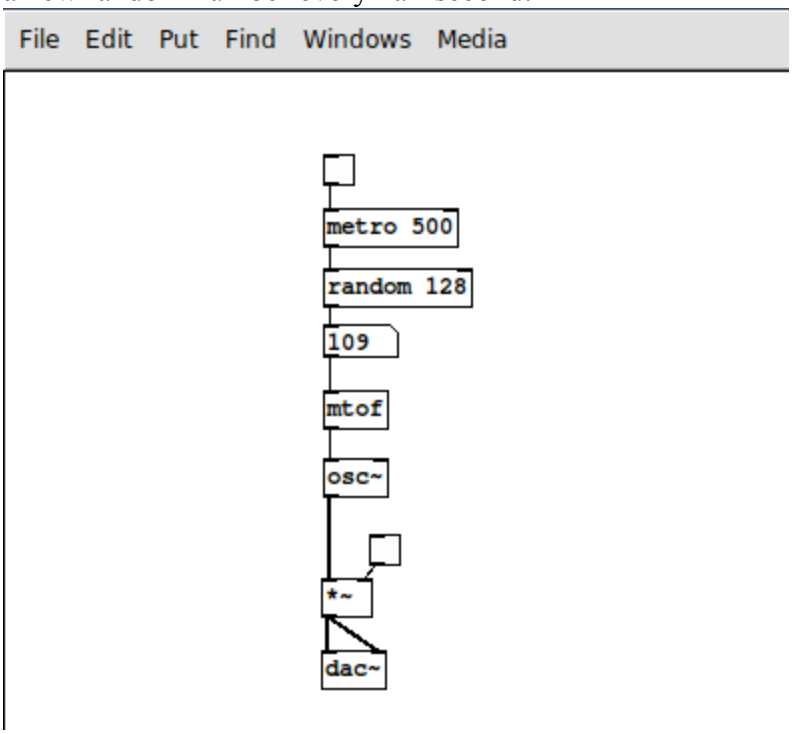


The user is recommended to make the following patch on their own to output random midi numbers 0-127 with bangs.



The bang object [bang], or GUI object as seen above, is often accepted input by other Pd objects that directs the object to perform some action immediately. In this case, a bang is used to output random numbers.

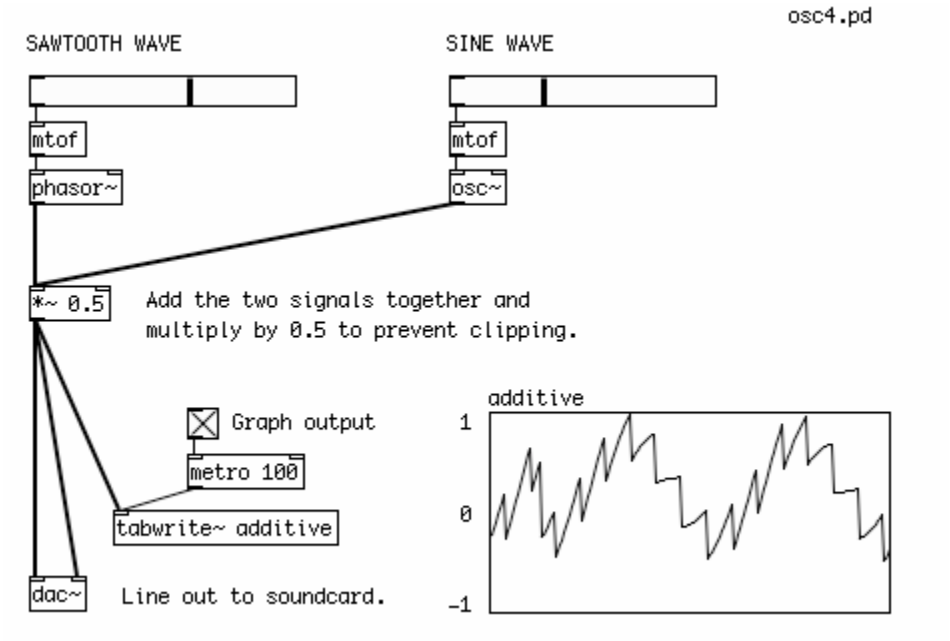
Another useful object in Pd is a metronome. This object has a timer that sends a bang every x millisecond (default measure of time in Pd). The patch below shows a metronome being used to output a new random number every half second:



Also note the toggle to goes to the second inlet of the *~ (signal multiply function). Toggles send a 1 or 0 when clicked (for on and off), thus when the toggle is clicked, a 0 is sent to the multiplier and multiplied with the signal, thus muting the music. Switching the toggle back turns the music back on.

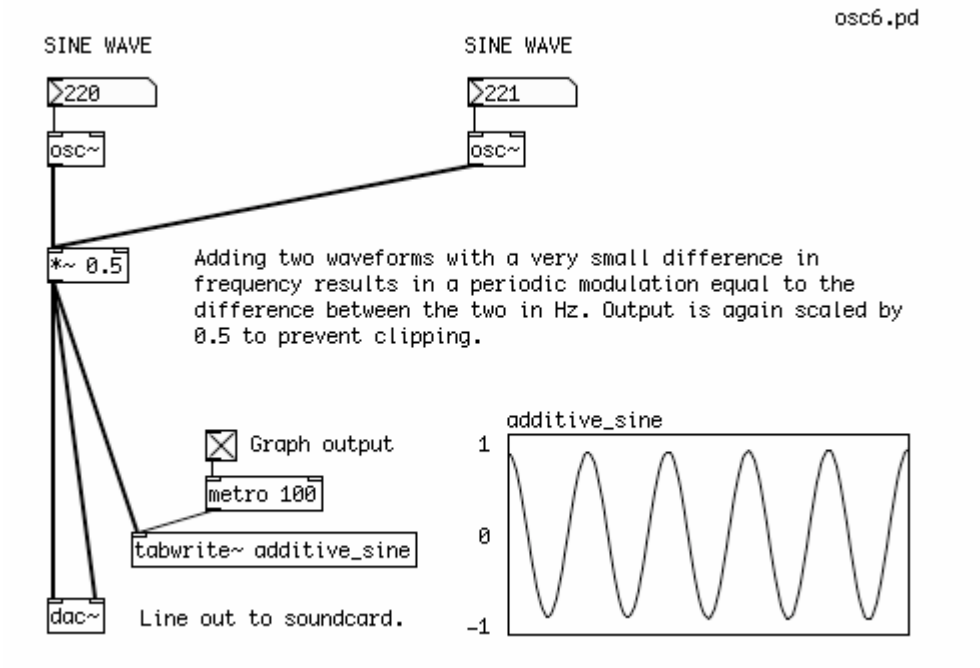
Exercise 3: Additive Synthesis

Because Pd adds the audio signals that come to the inlet of any audio object, combining two or more signals into a single waveform is simple. Adding a sawtooth wave and a sine wave results in the following waveform:



It is import to note here that the combined signals are multiplied by .5, since any signal outside the range of -1 and 1 will be clipped, which often causes distortion and very nasty sounds.

If you combine two waveforms whose frequencies are very close to each other, the combined values of the two waves interfere with each other, causing a periodic modulation of the sound. The frequency of this modulation is equal to the difference of the two original frequencies, in Hz. This is known as a "beating frequency," or "phase interference." The sound of two oscillators slightly de-tuned from each other is often used for different kinds of electronic music sounds, such as a "fat" bass effect. The following example shows this:

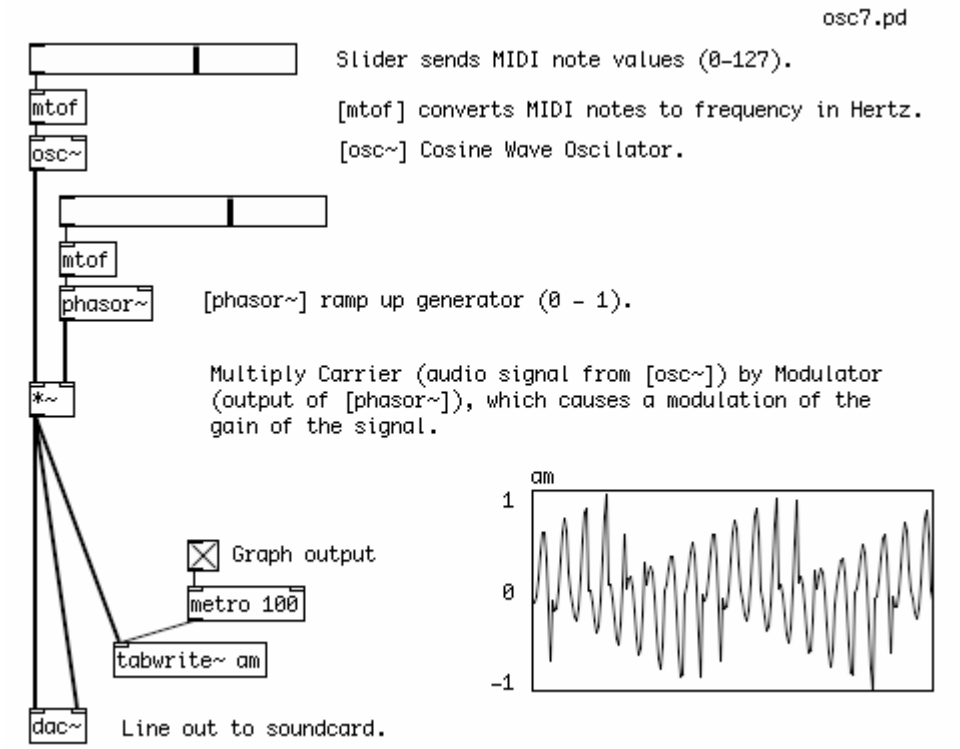


This is just one technique for combining waves in a meaningful manner. The next section described different modulation techniques.

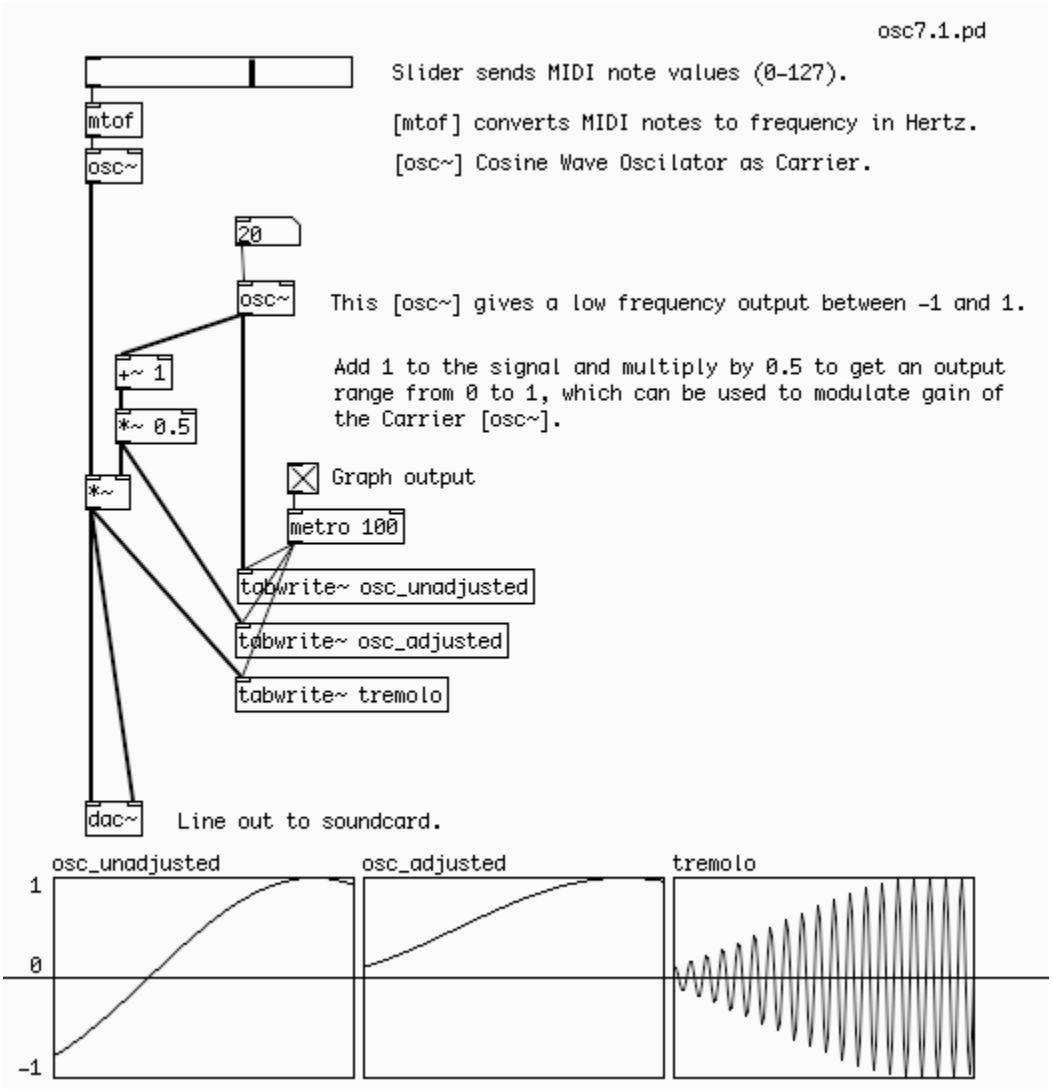
Exercise 4: Amplitude Modulation

Amplitude Modulation Synthesis is a type of sound synthesis where the gain of one signal is controlled, or modulated, by the gain of another signal. The signal whose gain is being modulated is called the "carrier", and the signal responsible for the modulation is called the "modulator". In classical Amplitude Modulation, or AM Synthesis, both the modulator and the carrier are oscillators. However, the carrier can also be another kind of signal, such as an instrument or vocal input. Amplitude Modulation using a very low frequency modulator is known as Tremolo, and the use of one audio signal to Amplitude Modulate another audio signal is known as Ring Modulation.

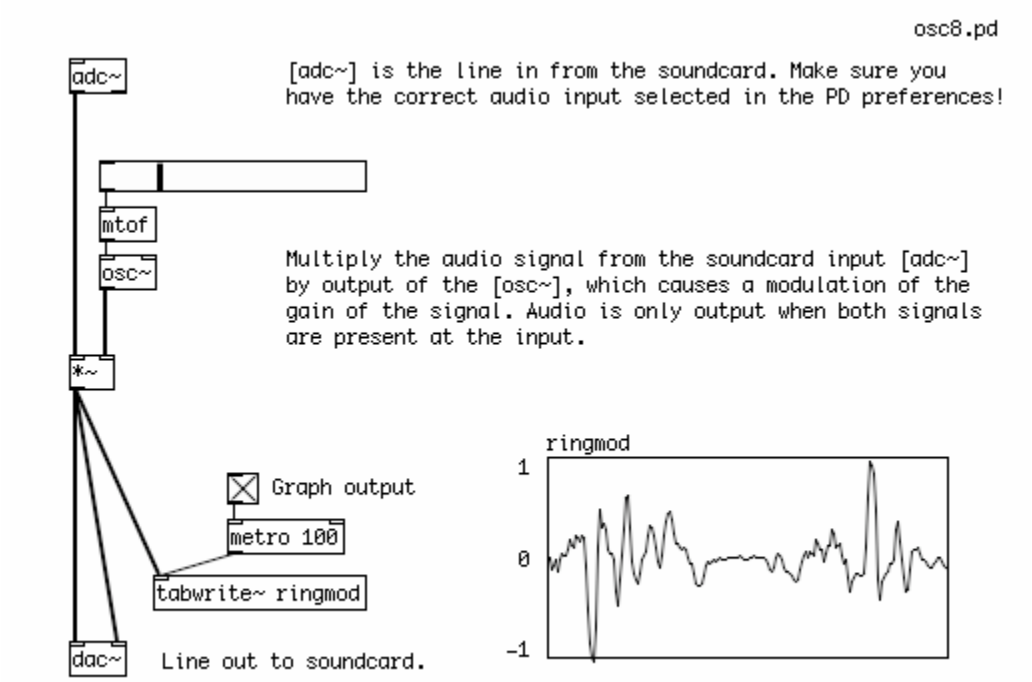
Classical AM Synthesis is created by using one oscillator to modulate the gain of another oscillator. Because we are changing the gain of the carrier oscillator from 0 (no gain) to 1 (full gain), the modulating oscillator must output a signal which changes between 0 and 1. This is most often done at audio frequency rates from 20 Hz and up. In this case, the sawtooth waveform of a [phasor~] is used as the modulator, and the sine waveform of an [osc~] is the carrier.



Tremolo is a form of Amplitude Modulation where the gain of an audio signal is changed at a very slow rate, often at a frequency below the range of hearing (approximately 20 Hz). This effect is commonly used to alter the sound of organs or electric guitar. Since a sine wave is often used for a smooth-sounding tremolo effect, in this patch we have taken the output of an [osc~], which normally moves between -1 and 1, and scaled it so that its output is now from 0 to 1. This is known as adding a DC Offset to the signal.



You can also modulate one audio signal with another audio signal (i.e. a signal which has both positive and negative values). This effect is called Ring Modulation. If you have a microphone connected to your computer, try the following patch. The sound of your voice will enter Pd through the Analog to Digital Converter [adc~] object (the line in from the soundcard), and be modulated by the sine wave of a [phasor~] object. Notice that there is no sound when only one audio signal is present (i.e. when you are not speaking). This is because one audio signal multiplied by zero (no audiosignal) will always be zero. And the louder the input signal is, the louder the output will be.



* The example can often cause a feedback loop from the speakers to the microphone. In order to prevent this, try wearing headphones!

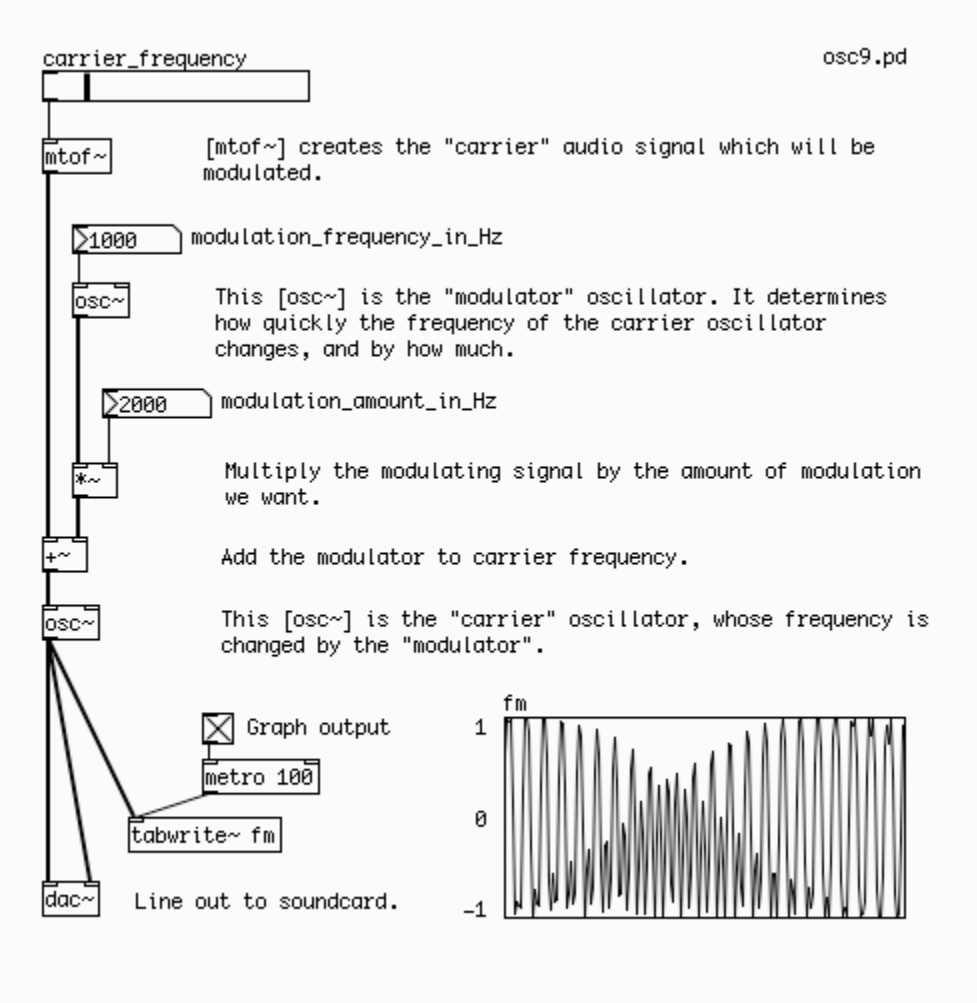
Exercise 5: Frequency Modulation

While Amplitude Modulation Synthesis changes the gain or volume of an audio signal, Frequency Modulation Synthesis, or FM Synthesis, is used to make periodic changes to the frequency of an oscillator. In its simplest form, Frequency Modulation uses two oscillators. The first is the carrier oscillator, which is the one whose frequency will be changed over time. The second is the modulator oscillator, which will change the frequency of the carrier.

For the carrier, we only set the base carrier frequency using a Number box and a MIDI to Frequency [mtof~] object. Because all the adjustments afterwards will be done by audio signals, it's best to use the audio version of [mtof], hence the tilde is added to its name.

The modulator is where we do most of the adjustments. The first thing we want to do is set the frequency of the modulator, i.e. how fast it will change the frequency of the carrier. We do this with a Number box. The second thing we want to set is how much change we will make in the base frequency of the carrier. So the output of the modulator [osc~] is multiplied by another Number box using an Audio Multiplier [*~] object to get the modulation amount.

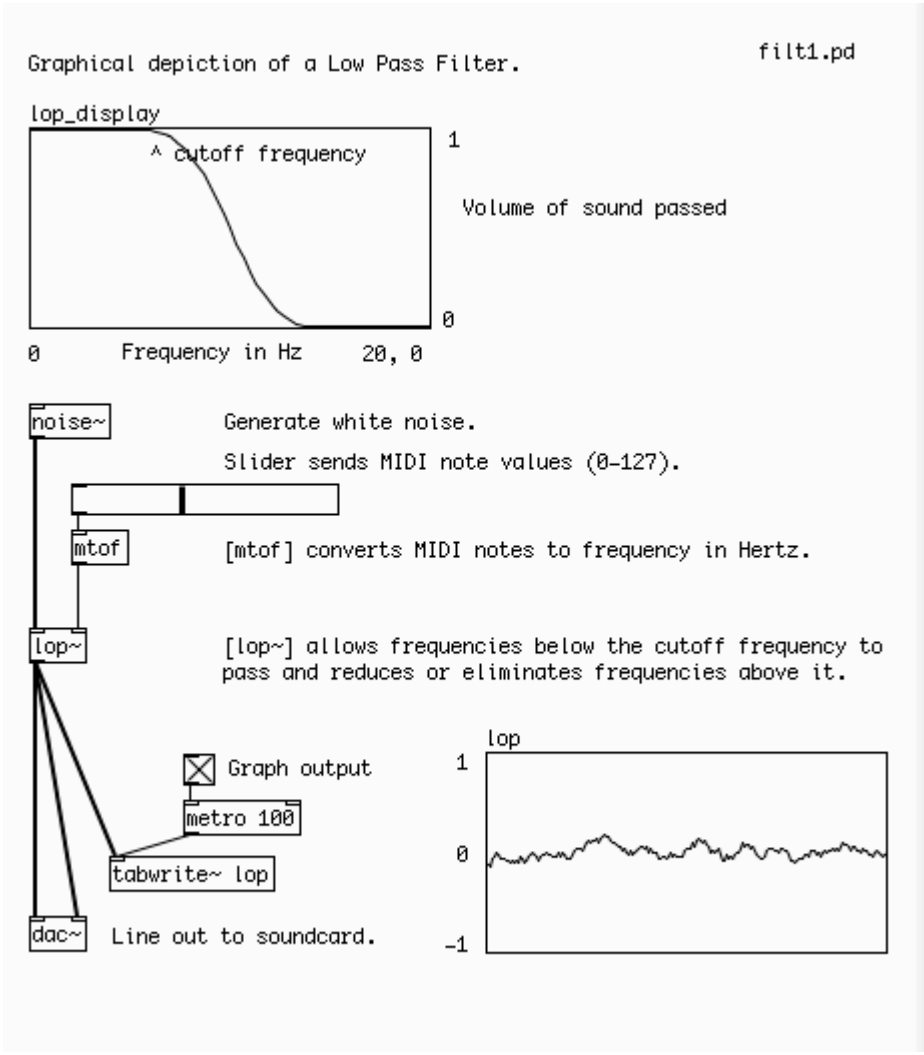
When this stream of numbers, which is changing with the speed the modulator and in the range set by the modulation amount, is added to the carrier frequency, then the carrier frequency will change as well. This stream of numbers is sent to the second [osc~], where it produces a complex sound which you can see in the graph.



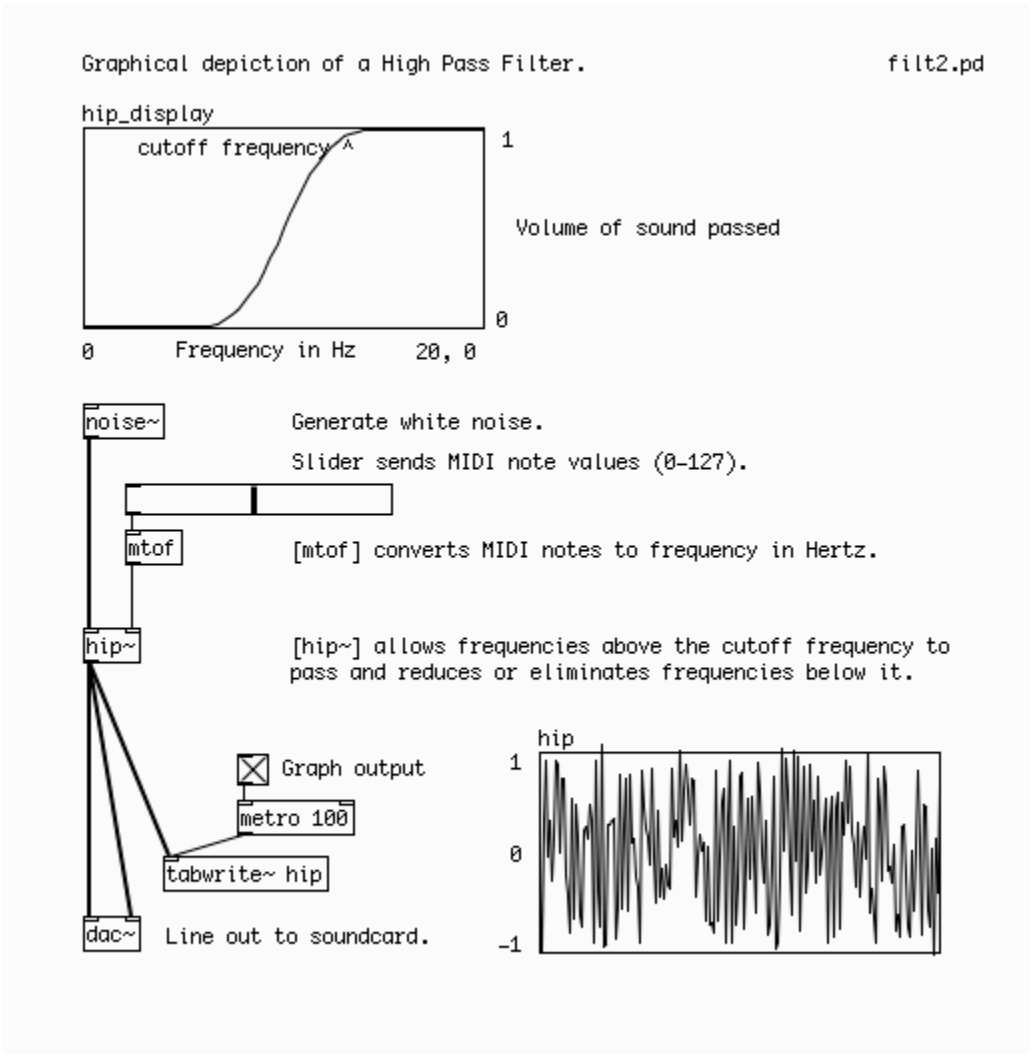
Exercise 6: Filtering

A filter works by allowing some frequencies through, while reducing or eliminating others. There are three different kinds of basic filters to work with in Pd: the Low Pass Filter, the High Pass Filter and the Band Pass Filter, as well as a Voltage Controlled Filter which can be controlled using some of the LFO techniques we have already looked at.

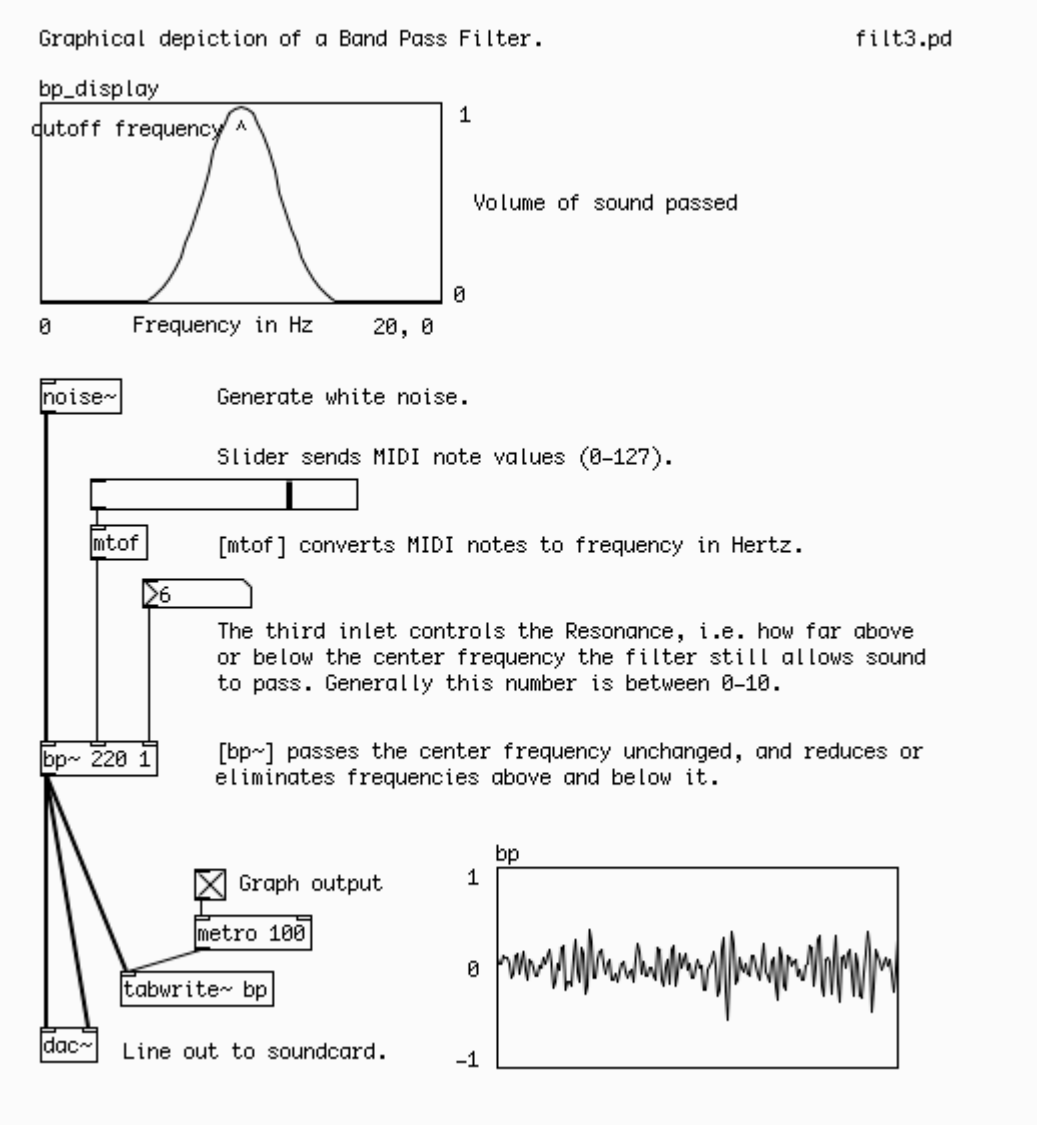
A filter which allows only low frequencies to pass is called a **Low Pass Filter**. The object for this kind of filter in Pd is [lop~]. It has one inlet for audio and one inlet for a number which determines the frequency in Hertz where the filter starts to reduce the audio (the Cutoff Frequency). Frequencies above the Cutoff Frequency are reduced or eliminated.



While one which allows only high frequencies is called a **High Pass Filter**. The object for this kind of filter in Pd is [hip~]. It has one inlet for audio and one inlet for the Cutoff Frequency. Frequencies below the Cutoff Frequency are reduced or eliminated.



A filter which allows some range of frequencies between highest and lowest is called a **Band Pass Filter.** The object for this kind of filter in Pd is [bp~]. It has one inlet for audio, a second inlet for the center frequency that it will allow to pass and a third inlet for the **Resonance**, which determines the width of the range of frequencies it allows to pass (the Pass Band). The Center Frequency will pass unchanged, and frequencies higher or lower than that will be reduced or eliminated. How much they will be eliminated depends on the Resonance. Useful numbers for the Resonance tend to be between 0 and 10.



13. Evaluation of learning achievement

None.

14. Glossary

None.

15. Additional useful links

None.

16. Contributors

a. Initial author: Conor Scott
b. Reviewers: Chris Brown, Seungwon Yang