DataPond

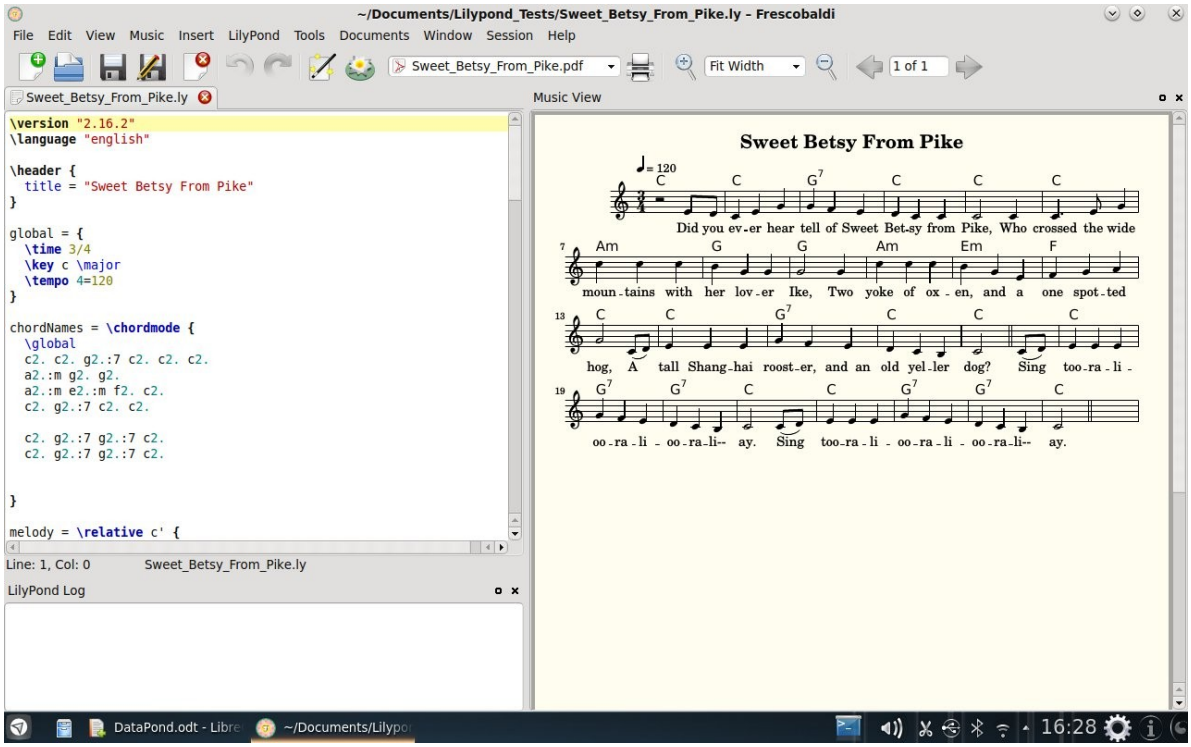
Data Driven Music Builder

Over the past couple of months I have been attempting to teach myself to read music. This is something that I have been meaning to do for a long time. In high school the music teachers seemed to have no interest in students other than those with existing natural talent. The rest of us, they just taught appreciation of Bob Dylan. I can't stand Bob Dylan.

So I never learned to read music or play an instrument. Having discovered the enjoyment of making stringed instruments such as the papermaché sitar, the Bad-Arse Mountain Dulcimer, a Windharp (inspired by this one), and a couple of other odd instruments. More recently I made a Lap Steel Slide guitar based on the electronics described in David Eric Nelson's great book "*Junkyard Jam Band"* and on Shane Speal's guide on building Lapsteels. I built mine out of waste wood palette wood and it connects to an old computer speaker system I pulled out of my junk box and turned to the purpose. It ended up sounding pretty good – to my ear at least. Anyway, I have instruments-a-plenty, but didn't know how to play'em so I decided to teach myself music.

The biggest early challenge was reading music. I'm sure it would have been easier if a mathematician had been involved in designing the notation system. To teach myself I decided that reading sheet music and transcribing it into the computer would be a good way of doing it.

Lilypond is a powerful sheet music "engraving" utility which has a powerful graphical front end called "*Frescobaldi*". Reading the music into Frescobaldi, I was able to press a button to get Lilypond to produce a pdf of the music notation I had written. If I was right, the pdf would match the original sheet music I had been reading. I quickly found that Lilypond had another benefit; it produced MIDI files at the same time. Now I could hear what the music was supposed to sound like if I processed the MIDI through LMMS, a digital audio workstation.



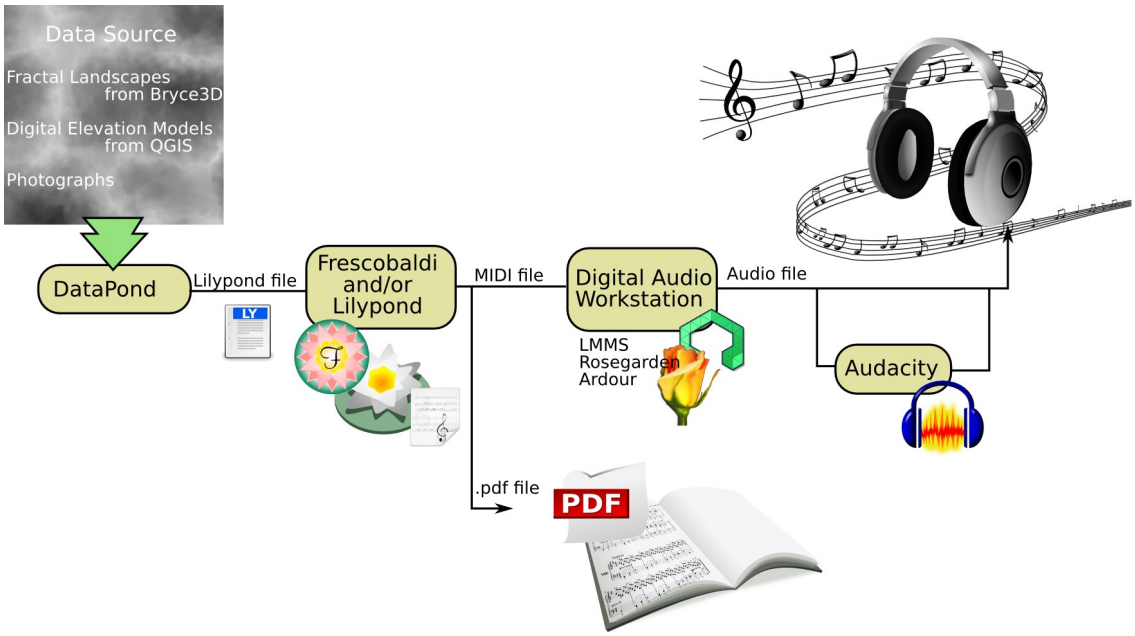
Frescobaldi Interface - the left panel has the Lilypond Code

It occurred to me that here was an interesting opportunity to produce procedural music in sheet form and as MIDI. All I needed to do was write a program to output a Lilypond file, which is just text. And so the idea for *DataPond* was conceived.

What is DataPond?

DataPond is a python script that takes a data source and processes it into a Lilypond text file. The data source has been chosen to be an image. DataPond uses the image to extract a profile from a line across the image defined by the user.

DataPond will not produce music itself, the output file needs to be processed my Lilypond to produce a pdf of the sheet music and a MIDI file. The MIDI file can be processed in any Digital Audio Workstation that will render it into an audio file capable of being played on any media player.



The DataPond Tool-Chain from Data to Audio File

How DataPond Works

Think of it a bit like an assembly line where the building blocks are riffs. The riffs are created, and then combined to make the song. Along the way musical rules are applied and the data is used to shape the music. Chords rather than melody provide the underlying framework for the construction of the song.

Ignoring the song name generator, here is a description of the process.

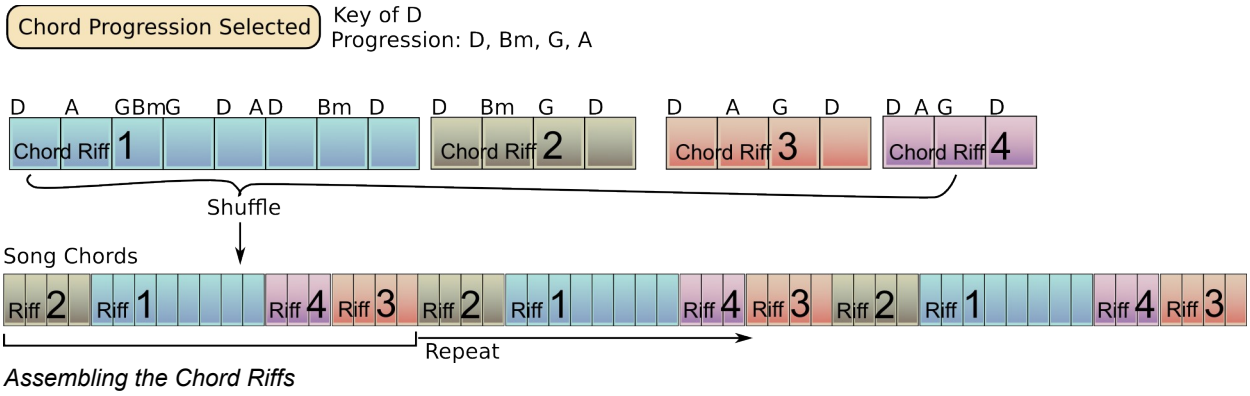
Score Time and Key
The time and key of the music is either selected by the user or randomly chosen. This determines the scale that will be used for the melody, the chord progressions, and the notes making up the chords within that key.

Chord Riff Number and Size
The lengths of the chord riffs are determined, based on the number of riffs selected by the user, the target length of the score (guideline only), and the desired number of repeats.

Chord Riff Timing
Within each riff the times for which each chord plays is chosen using a weighted random selection.

Chord Riffs
Using the chord progressions dictated by the song's key, the chord riffs can be populated by the Lilypond notation for the chords and their timing.
At the same time another series of riffs are built that match the chord riffs but capture only the first chord in each bar. This is used later when building the song melody.

Song Chords
The song chords are just the selection of available chord riffs shuffled up and then repeated to fill out the song to the target song length.
In parallel to this the riffs capturing the first chord in each bar are assembled in the same order as the song chord riffs.

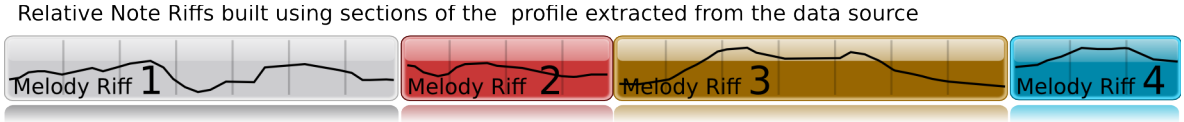


Assembling the Chord Riffs

Melody Riff Number and Length
In a similar manner to the chord riffs, the number and length of the melody riffs is determined based on the target length of the song, the user defined maximum size for the melody riffs and the desired number of repeats.

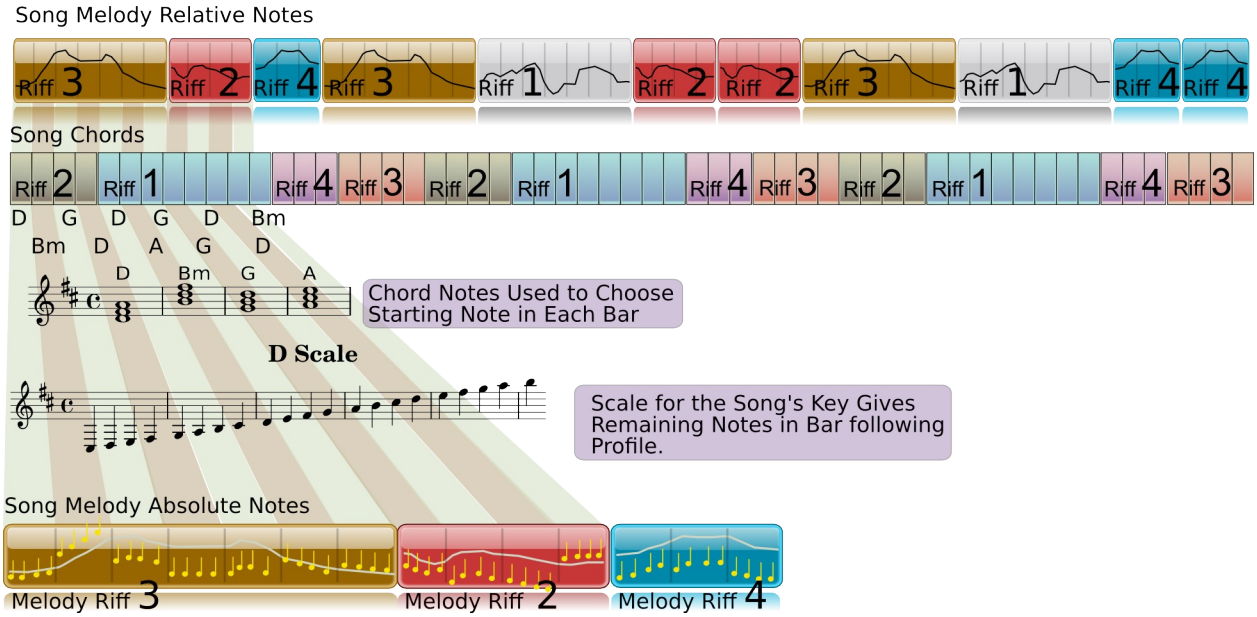
Melody Riffs – Note Timing
Again, like the chord riffs, each melody riff note timing is governed by probability tables which influence the random choice of note length. This probability table for the first note note in each bar is different from the probability table for the notes in the rest of the bar. This allows the user to influence the length of the first note in each bar.

Melody Riffs – Relative Notes
The melody riffs note positions developed in the previous step are populated by a number that describes the numbers of notes to rise or fall relative to each other. This number comes from a subsample taken from the data. Essentially each riff then consists of instructions that are ;“up a note, up two notes, stay at this note, down a note, down a note etc.”



Song Melody - Relative Notes
The various melody riffs containing the relative notes are then combined into a song melody which is entirely expressed in “up a bit, down a bit” to reflect the segments of data profile each melody riff contains.

Song Melody – Absolute Notes
The diagram below attempts to explain the process of creating the actual melody expressed in notes.



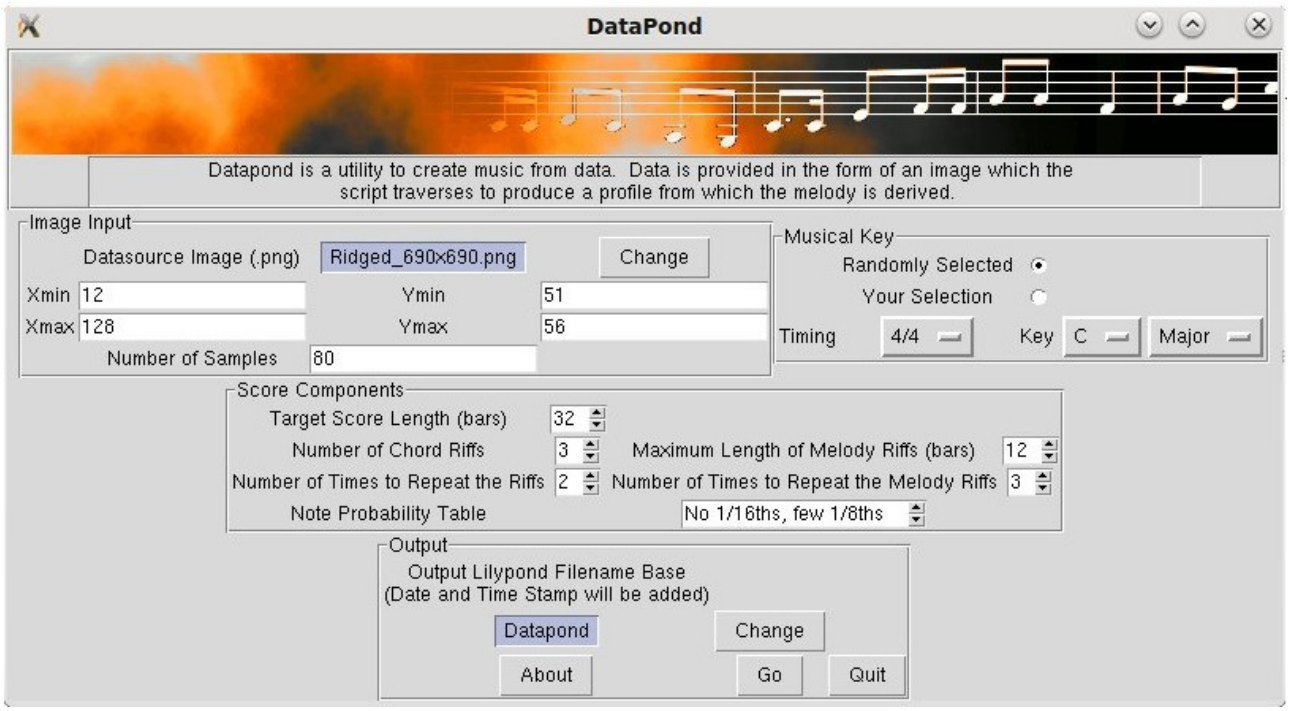
There are four components that influence the song melody;

- The song chords – the first chord in each bar.
- The notes making up the chord being played for a particular bar
- The scale for the song key. This limits the notes available to use.
- The song melody expressed as relative notes.

For each bar, DataPond looks at the first chord being played. The notes that make up the chord give it a choice of three notes from which to build the actual notes for the melody in that bar. The first note locks in the starting point for the rest of the notes. The actual notes in the rest of the bar can be found by going up and down the scale for the song's key as directed by the relative notes for that bar.
Now we have a melody consisting of the actual notes and note lengths. With these and the chords, we have a song.

How Do You Operate DataPond?

There are two options for running Datapond, using the pre-built binaries or using the Python code directly. The binaries use the interface shown below while the Python code can either be the GUI version or the command line engine on its own.



Using the Interface

The GUI interface is a little ugly, I know, but it works.

- **Image Input:** Use the “Change” button to select a different image. The images I have supplied all include their size in the file name to make it easier for the user to select appropriate coordinates (Xmin, Xmax, Ymin, and Ymax) for the sampling line. If you select coordinates outside of the image, it will result in an error. The number of samples refers to the number of points along the selected line to give the profile used in the melody riff processes.
- **Musical Key:** As the tile suggests, you can select the key and its timing or let the script randomly select it for you. Changing anything in the drop downs will automatically switch the selection to “your Selection”.
- **Score Components**: This is where the user selects the target total length of the score. This is a guideline only and will depend of the details of the melody riffs and the chord riffs. In this section you can define the number of chord riffs and how many repeats you want. You can also set the maximum desirable length in bars for the melody riffs and the number of melody riff repeats. You will need to be mindful how you set these so they can be accommodated within the desired score length. Errors will result if they are too wild. There are four note length probability tables to chose from. Hopefully the titles for these make sense.
- **Output**: This is where the output filename can be defined. There is a small bug here in that you can navigate to any directory you like, but it will produce your file in the “Output” directory. Sorry about that. The filename you define will have the date and time appended. This is so you can hit “Go” a number of times to generate a bunch of Lilypond files without having to change the output filename and without accidentally overwriting your latest creation.
- **“Go”** is the button that makes it all happen. It will pop up a window that reports a Lilypond file has been created. If you were to look in the Outputs directory you will find your file ready for processing through *Frescobaldi* and/or *Lilypond*.

The DataPond engine is the python3 code command line version which will allow a user to get at the chord probabilities and many other bits that can be tweaked for better results. Enterprising users will be able to hook in other data sources with little difficulty.

Diving into the Code

If delving into the code itself, the useful parts to look at are:

LoadDefaultsFlag
Around line 30, there is a `LoadDefaultsFlag` which will load up a number of preset entries that would otherwise be filled with a series of questions at the python prompt. The pre-set entries can be found in the "User Input " section starting at line 490.

The note length probability tables.
In the DataPond engine python code they are in the lines starting around 375. Each set contains two dictionaries; `NoteLengthFirstProbSetX`, and `NoteLengthGenProbSetX`. One for the probabilities of the first note in the bar being a particular length, and the other contains the note length probabilities of the remaining notes in the bar. Refer to the dictionary (`NoteLengths`) at about line 368 which contains the note lengths being referred to where the note lengths are expressed as decimal versions of their fractions (ie a quarter note is 0.25).

Starting at line 439 is a set of tables matching the note length probability tables. This contains the length of the smallest note used allowed in the note length probability tables and a number that is slightly bigger than that note length. For instance where we have allowed 1/16th notes, the smallest note will be 0.0625 and the threshold we have defined to be slightly bigger at 0.07. To explain why this is needed; the process of allocating note lengths to a particular bar adds notes until there is either no lime left or a length of time that is under the threshold defined above. If the remaining time in the bar is smaller than this value and not zero then it must be the smallest note is the only one that can fit there. Having written that I suspect this condition is probably never arrived at anyway, because the probability of the getting the note of the correct size will for that last note in the bar will be 1.

Chord Lengths and Their Probabilities.
The chord lengths are set to be no shorter than a quarter note, but your may with for shorter ones. In this case you will need to add some entries into the `ChordLengths` dictionary around line 476.

As with the melody notes, the chord lengths also have a probability associated with them. This dictionary is `ChordLengthsProbWt` and can be found around line 480.

The chord progressions are also governed by probability tables which can be found in lines 485 and 486. The probabilities are different for major and minor keys. Each entries refers to the probability of a particular chord within the progression being played.

Data Profile Generation
This is done by analysing an image and taking value from it along a line defined by the user. This process is found in the section "Analyse the Image or data stream" starting around line 865. The output from this section that is used to produce the melody is the list `zHt`. So if you call any list of data points `zHt` and slot it in place of this section, you will be able to run DataPond on any data source.

What to watch out for

There are a few things to watch out for.

Lilypond version
When you produce your Lilypond file it will have the line;

```
\version "2.16.2"
```

This works for me because this in the version of Lilypond I have installed. Chances are you will have a later version and so you will need to change this to match your version, otherwise Lilypond will object.

You can either do this post DataPond using a text editor or if you are using the Python code look at line 39 in

the ***DataPond_Engine*** or line 420 in ***DataPondGUI*** and change this line to match the Lilypond version you are using.

```
            '\\version "2.16.2"\n\header {\n  '
```

Be sensible with your riff length choices
Datapond will pop up errors if you go overboard with the length of your riffs . If the total song chord bars and the total song melody bars can't match, an error will occur. I had to do a fudge to cater for the potential for a slight mismatch in the length of the song chord and the song melody. I don't like it, but it was needed to prevent the script from crashing when the user puts in silly lengths of riffs and repeats that make no sense given the desired length of the whole song. Where it has the “backstep” can be removed for smarter users.

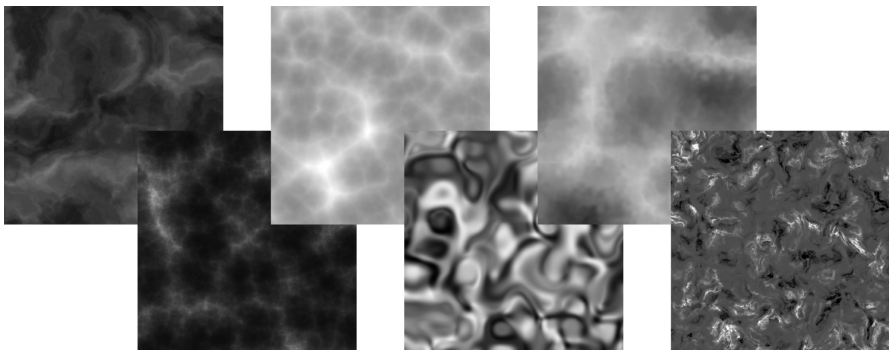
This program is not that tidy
It has not been made idiot proof. This is a program, not software. So expect it to occasionally spawn errors.

Getting images for DataPond to work on

I have included a small number of images for DataPond to work on, but here are some suggestions for other places you may like to find images. All images must be in ***png*** format. Where images have multiple colour and alpha channels I have set the program to only look at the first channel.

GIS: Height maps and digital elevation models as rasta images are great sources. You can find an excellent description of outputting height maps from QGIS in John Flower's site.
Bryce, Terragen, Vue: Bryce, Terragen, and Vue are fractal terrain generators. I am familiar with Bryce and know you can export some of the heightmaps from this. I presume the same can be done with Terragen and Vue.
Other fractal generators: Anything that produces an interestingly variable image should work.

Although I haven't tried it I should think normal maps and any photograph will work.



You can download a **bundle of height maps I created using Bryce from here.**. The zip file is 74Mb and so these images weren't included in the code or binary bundle.

Don't like the sound? Modify it.

DataPond does not output music as such. It is merely a tool along a tool-chain. The other members of that tool-chain are quite powerful and will allow you to modify the output from DataPond to something that suits your style and taste. Here are a couple of things you could do.

- Use Frescobaldi and Lilypond to change speed and transpose to other keys.
- Use LMMS, Rosegarden, or other Digital Audio Workstation to change the instrumentation and add extra stuff such as drum tracks.

The Code:

The code comes in several flavours.

Binaries
We have a version for Linux and for Windows. The Windows version was built on a 32Bit XP machine and so hopefully will work OK on any of the later Microsoft abominations. Both editions were built with cx_Freeze. Just unzip the files to where-ever you want to run them.

- **DataPond_Linux32.zip**

- **DataPond_Win32.zip**

Look for the ***DataPondGUIv4.exe*** in amongst the vast screeds of other stuff.

Source:
Both source packages have a small selection of images included. Both run under Python 3.
- **DataPondGUIv4.zip** – Tkinter interface together with the engine
- **DataPond_Enginev1.zip** – The engine with a python prompt interface (or less).

Hopefully the comments will be enough to explain what's going on. I try to choose descriptive names for the variables to help the process...Well actually it's mainly so that I know what it all means if I try read through the code again myself in the future.

Some examples of the output

There are several examples of the output that can be found on my website (http://www.techmonkeybusiness.com/datapond-data-driven-music-builder.html). Those that are just a piano are the MIDI processed through LMMS without any fiddling with the instruments. The others have been modified in LMMS by selecting other instruments from those available in the Fluidsynth Soundscape. All titles have been chosen by DataPond.

While reviewing these, I found my brain almost recognising patterns but then not quite finding them again through the rest of the tune. I guess this reflects the presence of relatively long riffs and the repeats. I feel I should try some shorter riffs and greater number of repeats to see if this makes it sound even more song-like.

So that's it. Enjoy.

www.techmonkeybusiness.com