What does a nonabelian group sound like?

Reginald Bain, Matthew Corley, William DeMeo

1 Short term goals

Choice of programming language and other software.

(wjd: December 15, 2013) This will be an ongoing discussion, but to get the ball rolling it seems a functional programming language like Scala would be ideal for expressing the math that we want to implement, so I suggest we start with that as our primary language and use Java as a fallback.

Incorporating finite groups into Scala or Java programs.

(wjd: December 15, 2013) One of our primary goals is to write a program to do convolution of signals defined on finite nonabelian groups, so we need to incorporate finite groups into our programs. The main thing we need is the multiplication table—that is, the table that tells us the result of multiplying any two group elements—as well as a function that tells us the inverse of a given element. We could write programs to do all this from scratch, especially for small groups that are easy to represent (like semidirect products of cyclic groups). However, there is plenty of code already written for finite groups, and we should avoid reinventing this wheel. I suggest the following strategy instead:

1. Use the GAP software for exploring various finite groups to identifying some groups we want to consider.

2. Use my GAP program gap2uacalc which writes GAP groups to XML files and represents them as general (universal) algebras—i.e., simply a set with operations defined on the set. (More details about this and the XML format used by gap2uacalc will appear below.)

3. To read the resulting XML files into our Scala or Java programs, we can use a UACalc Java class called AlgebraIO (see the UACalc javadoc). (The gap2uacalc program writes groups to a file in UACalc's XML format.)

Convolution background.

(wjd: December 15, 2013) The main operation/function we will considering is convolution, and we should think about how best to view this mathematically, as well as how best to represent it in the computer. In this section is some background on the mathematical aspects. In the next section are some initial thoughts on how to code this up in Scala.

Let $\mathbb{C}^G$ denote the set of complex valued functions defined on the group G. That is

$$\mathbb{C}^G = \{f : G \to \mathbb{C}\}.$$

(In the Tolimieri-An books, [6] [7], this set is also denoted by $\mathcal{L}(G)$.) If the group has $|G| = n$ elements, say, $G = \{x_0, x_1, \ldots, x_{n-1}\}$, then each function $f \in \mathbb{C}^G$ can be represented as a length-n vector in $\mathbb{C}^n$—namely, the vector of its values on G:

$$\mathbf{f} = [f(x_0), f(x_1), \ldots, f(x_{n-1})].$$

Given two functions f and g in $\mathbb{C}^G$, the *convolution of f and g*, denoted, $f * g$, is also function in $\mathbb{C}^G$ and is defined by the values it takes at each $x \in G$ as follows:

$$(f * g)(x) = \sum_{y \in G} f(y) g(y^{-1}x). \tag{1}$$

Note that this is a weighted sum of translations of g. Indeed, let $\mathsf{T}_y : \mathbb{C}^G \to \mathbb{C}^G$ denote the *translation by y* operator—that is, T_y maps a function $g \in \mathbb{C}^G$ to a translated version of itself, $\mathsf{T}_y(g)$, which is defined at each $x \in G$ by $\mathsf{T}_y(g)(x) = g(y^{-1}x)$. Then (1) can be written as

$$(f * g)(x) = \sum_{y \in G} f(y)\mathsf{T}_y(g)(x), \tag{2}$$

a sum of weighted translations of g where the coefficients $f(y)$ are the weights, and $\mathsf{T}_y(g)$ is the function g "shifted" by y. (When G is the abelian group $\mathbb{Z}/n\mathbb{Z}$ with addition modulo n, we have $\mathsf{T}_y(g)(x) = g(y^{-1}x) = g(x - y)$, so in this case $\mathsf{T}_y(g)$ is literally g shifted by y units to the right.)

Equation (2) defines the convolution, $f * g$, by giving its value at each $x \in G$. Using the translation operator, however, we can define convolution "functionally," instead of element-wise, as follows:

$$f * g = \sum_{y \in G} f(y)\mathsf{T}_y(g) \tag{3}$$

(Pause to look at the right hand side of (3), and let it sink in that this is a function that takes arguments $x \in G$; compare with the right hand side of (2).)

This is fine, but it is also useful to think of (3) as f acting on g. Indeed, on the right hand side of (3) we have the operator $\sum_{y \in G} f(y)\mathsf{T}_y$ that maps the function g to the function $f * g$. But on the left hand side we have a binary operation $f * g$, written in infix notation, which doesn't jibe very well with this functional interpretation. So, instead of saying "the convolution of f and g", and writing $f * g$, we will say "the convolution **by** f **of** g," and write $\mathsf{C}(f)(g)$. In this way, we have the *convolution by f* operator:

$$\mathsf{C}(f) = \sum_{y \in G} f(y)\mathsf{T}_y, \tag{4}$$

which is a weighted sum of translation operators. The function $\mathsf{C}(f)$ takes other functions, like g, as its argument.

So, the functional types we have here are the following:

$$\mathsf{C} : \mathbb{C}^G \to (\mathbb{C}^G)^{\mathbb{C}^G}$$

Given $f \in \mathbb{C}^G$,

$$\mathsf{C}(f) : \mathbb{C}^G \to \mathbb{C}^G$$

Given $f \in \mathbb{C}^G$ and $g \in \mathbb{C}^G$,

$$\mathsf{C}(f)(g) : G \to \mathbb{C}$$

Or, in the notational style of a functional programming language like Scala:

$$\mathsf{C} : (G \Rightarrow \mathbb{C}) \Rightarrow ((G \Rightarrow \mathbb{C}) \Rightarrow (G \Rightarrow \mathbb{C}))$$

Given $f \in \mathbb{C}^G$,

$$\mathsf{C}(f) : (G \Rightarrow \mathbb{C}) \Rightarrow (G \Rightarrow \mathbb{C})$$

Given $f \in \mathbb{C}^G$ and $g \in \mathbb{C}^G$,

$$\mathsf{C}(f)(g) : (G \Rightarrow \mathbb{C})$$

Initial thoughts on implementing convolution in Scala.

(wjd: December 15, 2013) Functions in Scala can have multiple argument lists. If we have, say, 4 argument lists, then we can write

```
def f(args1)(args2)(args3)(args4) = E
```

where `E` is some expression involving the arguments in the lists. This is equivalent to

```
def f(args1)(args2)(args3) = (args4 => E)
```

because `f(args1)(args2)(args3)` can be thought of as a function that takes as input the last argument list, `args4`, and returns `E`. Another way to write this is

```
def f = (args1 => (args2 => (args3 => (args4 => E) ) ) )
```

(This is called *currying*, named after one of its proponents, Haskell Curry.)

Here's what the interface to a convolution function might look like in Scala:

```
def convolution(f: Int => Double)(g: Int => Double)(x: Int): Double = {

    // insert convolution algorithm

}
```

This expresses precisely the functional view of convolution described mathematically in the previous. Given this interface, for each function $f \in \mathbb{C}^G$ we could define *convolution by* f as

```
def Cf = convolution(f)_
```

(The underscore is required in Scala when you leave off argument lists.)

Then, given another function $g \in \mathbb{C}^G$, we could define

```
def fg = Cf(g)
```

Thus, we have *convolution by* f *of* g defined functionally, rather than pointwise. Of course, we can evaluate this function at various points: `fg(x)`.

Alternatively, it might make more sense to take an object oriented approach (let's discuss). For example, it probably makes sense to define a CG class to represent complex valued functions defined on groups, and define the binary operation * for this class to be convolution of two CG objects. This can also be done very simply and elegantly in Scala.

Perhaps we should implement convolution both ways—I don't think it will be very difficult—and then use whichever seems more natural in a given context. In Tolimieri-An, there is a nice discussion of the mathematics of these two alternative views—i.e., convolution as a binary operation defined on $\mathcal{L}(G)$, versus convolution as a functional defined on $\mathbb{C}G$. (Note: $\mathcal{L}(G)$ and $\mathbb{C}G$ are simply two mathematical representations of the same object, that is, the set of complex valued functions defined on the group G.)

We should also discuss whether we will need to use complex valued functions, or if real valued functions will suffice.

A Excerpts from Original Proposal

Abstract. Underlying many digital signal processing (DSP) algorithms, in particular those used for digital audio filters, is the convolution operation. This operation acts on a signal, $f(x)$, and can be viewed as a weighted sum of translations, $f(x-y)$. Most classical results of DSP are easily and elegantly derived if we define our functions on $\mathbb{Z}/n$, the abelian (or commutative) group of integers modulo n (see [6]). The term *abelian* here refers to the fact that the basic group operation is addition (modulo n) which is a commutative operation (i.e., $x+y=y+x$).

If we replace this "index set" (the set on which functions are defined) with a *nonabelian* group—where the group operation is now multiplication, xy—then instead of the usual translation, $f(x-y)$, we have a *generalized translation*, $f(y^{-1}x)$. If we carry out convolution using this generalized translation, the resulting audio filters will naturally produce different effects than those obtained with ordinary (abelian group) convolution.

Dr. DeMeo, initiated research based on these ideas in 2004 and presented some preliminary findings at the International Symposium on Musical Acoustics (see [4], which received a "best paper" award). Similar ideas have been successfully applied to two dimensional image data as well as to other areas of engineering (see [1] and [7]). However, to date the application of nonabelian groups to audio signal processing seems relatively unexplored, and there are a number of fundamental open questions in this area that we hope to answer.

Research question. If the underlying index set of a digital audio filtering algorithm is modified to use various nonabelian groups (instead of the commonly used abelian group), how does this change the behavior of the filter and the resulting audio output?

Project time-line.
October 2013–December 2013: Become more familiar with music analysis/synthesis and DSP algorithms, and gain further knowledge of group theory and its role in classical DSP implementations.

January 2014–April 2014: Write code to implement algorithms for general nonabelian group DSP. Identify specific characteristics of groups that make them more (or less) useful as an index set on which to define DSP operations like convolution.

May 2014–October 2014: Gather and analyze results, and write up reports. Submit manuscript to an academic journal. Prepare for and attend conferences.

Project goals and objectives. We propose to explore the idea of using the underlying finite group (i.e., the index set) as an adjustable parameter of a digital audio filter. By listening to samples produced using various nonabelian groups, we hope to get a sense of the "acoustical characters" of finite groups. We will attempt to associate these acoustical features with various mathematical properties of the groups, and develop a classification scheme that might be useful to practitioners in audio signal processing and computer music composition.

- *Goals:* Develop the mathematical theory necessary to provide sonic characterizations of nonabelian groups. Discover which mathematical features of a group can be used to describe how a given DSP algorithm based on that group will behave. Produce computer software that allows users to process and manipulate musical signals using nonabelian group filters.

- *Objective 1:* Develop an understanding of the basic math underlying signal processing algorithms in general and convolution in particular and show mathematically what effects the use of a nonabelian group index set will have on the convolution operation.
- *Objective 2:* Find a short list of nonabelian groups that are useful for nonabelian group audio filters and effects processors, prove their effectiveness both mathematically and experimentally, and document these discoveries.
- *Objective 3:* Implement a software program that takes an audio signal as input and allows the user to apply filters corresponding to specific nonabelian groups to achieve different effects.

Methodology. We will conduct controlled experiments with very simple sound signals at first (sine waves and linear chirps), and filter these signals using standard convolution. Then we will filter the original signals using a generalized (nonabelian) convolution, substituting the underlying index set with various groups from the wide variety of nonabelian groups available in the SmallGroups library of GAP [5].

When we replace the index set $\mathbb{Z}/n$ with various finite nonabelian groups, in the beginning, the simplest examples of nonabelian groups (such as semidirect product groups), will be constructed "by hand" using GAP's `SemidirectProduct` function. Groups with a more complicated structure will be selected from GAP's vast SmallGroups library using various selection criteria. For each of the groups tested, we will implement the convolution function using the generalized (nonabelian) translation $f(y^{-1}x)$ in place of ordinary translation $f(x-y)$ used in classical convolution.

After completing these controlled experiments, we will analyze the results to compare the effects of the choice of group on the resulting convolution filter. Finally, we will attempt to make a connection between the mathematical properties of the group and the acoustical properties of the resulting convolution.

Both GAP and Matlab will be used for much of the initial prototyping and testing. Matlab provides easy methods for constructing wav files "from scratch" with its `wavwrite()` function. Additionally, Myoung An (a colleague of Dr. DeMeo) has provided us with the Matlab code that she and Richard Tolimieri developed for their work in image processing, where they applied nonabelian group filters to the processing of 2D digital images. This code will be a valuable resource as we seek to apply similar ideas to audio signal processing.

As the project progresses, we will likely use the JavaSound library and implement our generalized DSP algorithms in Java. JavaSound provides methods for reading and altering wav files frequencies and sound intensity levels, which will prove useful when we apply our generalized DSP algorithms to more complex sounds.

Anticipated results, final products, and dissemination. By the end of the Spring 2014 semester, I expect to have written Matlab programs to test the results of the modified DSP implementations described above. I also expect to have developed a Java software program which allows easy application of nonabelian group filters through a graphical user interface. I hope that the results will prove interesting and have practical applications for computer music composition.

The abstract for this project has already been accepted for presentation at the Joint Mathematics Meetings in Baltimore in 2014. In addition, I will submit the work to the International Computer Music Conference[1] (ICMC), the International Symposium on Musical Acoustics[2] (ISMA),

[1] http://www.computermusic.org/page/23/
[2] http://isma.univ-lemans.fr/en/index.html

and the 14th International Conference on New Interfaces for Musical Expression[3] (NIME). Previous work on topics related to this proposal by my faculty mentors have been accepted at both ICMC and ISMA, so we have high expectations for this project. I will write up a formal article describing the research and results and submit the manuscript to at least one scholarly journal in mathematics or music. Finally, if my project proposal is accepted and I become a Magellan Scholar, I will be honored to present the work at Discovery Day 2014.

List of Acronyms

GAP Group, Algorithms, Programming[4]

XML Extensible Markup Language

ICMC International Computer Music Conference[5]

ISMA International Symposium on Musical Acoustics[6]

NIME New Interfaces for Musical Expression[7]

DSP digital signal processing

References

[1] Gregory S. Chirikjian and Alexander B. Kyatkin. *Engineering Applications of Noncommutative Harmonic Analysis: With Emphasis on Rotation and Motion Groups*. CRC Press, 2002.

[2] J. H. Conway, R. T. Curtis, R. A. Wilson, S. P. Norton, and R. A. Parker. *ATLAS of Finite Groups*. Oxford University Press, 1986.

[3] William DeMeo. Characterizing musical signals with Wigner-Ville interferences. In *Proceedings of the International Computer Music Conference*. ICMC, 2002. Available from: http://math.hawaii.edu/~williamdemeo/ICMC2002.pdf.

[4] William DeMeo. Topics in nonabelian harmonic analysis and DSP applications. In *Proceedings of the International Symposium on Musical Acoustics*. ISMA, 2004. Available from: http://math.hawaii.edu/~williamdemeo/ISMA2004.pdf.

[5] The GAP Group. *GAP – Groups, Algorithms, and Programming, Ver. 4.4.12*, 2008. Available from: http://www.gap-system.org.

[6] Richard Tolimieri and Myoung An. *Time-Frequency Representations*. Birkhäuser, Boston, 1998.

[7] Richard Tolimieri and Myoung An. *Group Filters and Image Processing*. Kluwer Acad., 2004. Available from: http://prometheus-us.com/asi/algebra2003/papers/tolimieri.pdf.

[3] http://www.nime.org/nime2014/
[4] http://gap-system.org/
[5] http://www.computermusic.org/page/23/
[6] http://isma.univ-lemans.fr/en/index.html
[7] http://www.nime.org/nime2014/