A SPIN ON STOCHASTIC MUSIC (WIP)

CARLOS CORONADO

ABSTRACT. A spinner in its usual sense uses a stochastic process to pick an element of itself. Using elementary music structures, we are able to create works of stochastic music that can give light to the process from which they were made. The use of Python in SAGE is required to produce these, and SuperCollider is then used to hear those songs.

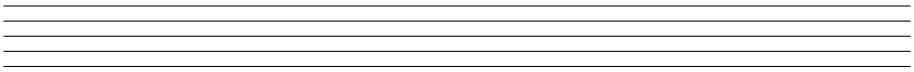
CONTENTS

1. Useful Music Theory

When discussing mathematical papers, often a musical background is not entirely necessary. For this specific one, we will introduce some elementary music theory useful to understand.

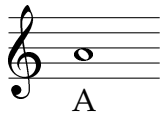
There are two main characteristics of music: notes being played and their duration. Duration of notes is sometimes referred to as rhythmic dictation, and for the majority of the paper, will be unused.

Musical notes are often written on a staff,

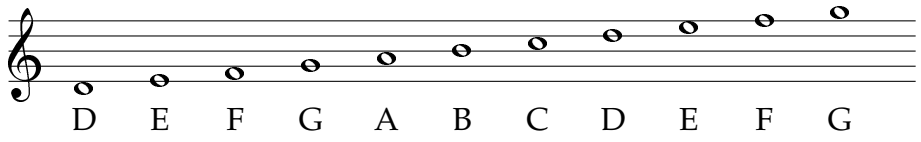


and are given name assignations through the help of a clef. Examples of clefs include the treble clef and bass clef. The treble clef is often used for instruments of higher pitch (frequency), while the bass clef is recommended for instruments of lower pitch.

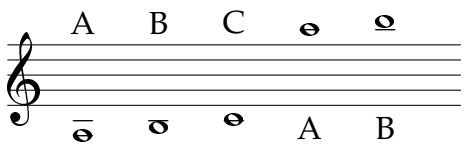
To visualize where the pitch of a note is on a staff, the whole note symbol, 𝅝 is used. All notes on that line will correspond to that same note unless otherwise specified by other clefs.



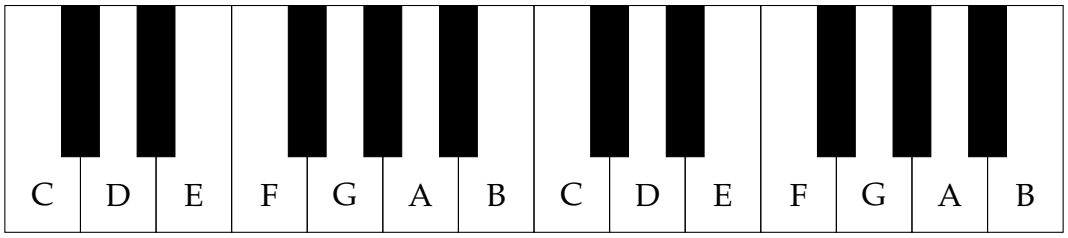
Musical notes are written on both the lines and the spaces between those lines, where higher pitches correspond to notes above lower pitches in the staff. The musical note A is of lower pitch than B, and so B is notated on the space directly above A. Then C is higher than B, and so on. Additionally, since notes are cyclical, G is below A. Thus,



Ledger lines are used for notes above or below the staff:



The notes we've looked at so far correspond to the white keys on a keyboard:



The black keys on the piano are notated with sharps ♯ and flats ♭. As an example, the black key in between the notes C and D is called either C♯ or its enharmonic D♭.

Steps are measured as distances between two consecutive notes on a piano. A half step is the distance between any note and its previous or the next key. For example, C to C♯ are a half step away from each other. A whole step is the distance between any key and two half steps in the same direction. For example, C to D.

An interval measures the space between two notes. The following graph can provide a better idea:

Base	C	C	C	C	C	C	C	C
Next Note	C	D	E	F	G	A	B	C
Interval	First	Second	Third	Fourth	Fifth	Sixth	Seventh	Octave

There are twelve notes in between the octave in the usual equal temperament system. A scale is a set of these notes arranged in a strictly increasing sequence, where the infimum is the note of lowest pitch and the supremum is the highest pitch.

The following scales will be widely used in the process of creating stochastic music: major, natural minor, harmonic minor, descending melodic minor, major pentatonic, minor pentatonic, whole tone, blues, triads, and chromatic.

Chords are notes played simultaneously from a scale. The use of triads and power chords will be extensively used in this paper.

An arpeggio is a increasing three note sequence from a chord. We will loosen this definition to be any increasing sequence of notes.

A highly used theoretical concept in classical music, especially in that of the Baroque period, is the use of chord progressions. These can be thought of as the easiest way to part-write music for soprano, alto, tenor, and bass voices.

2. SPINNERS

Intuitively, a spinner is comprised of two parts: a set of elements arranged in a circle and an arrow that chooses one of these elements.

Definition 2.1. *Let $\mathcal{C}$ be a finite strict weak ordered set. Also let $X = \{X_0, X_1, ..., X_n\}$ be a set of independently and identically distributed (i.i.d.) random variables. A **spinner** is a function $S : X \to \mathcal{C}$, that we can define recursively or by partial sums, both of which are received after a modulo.*

The reader can think of $\mathcal{C}$ as their favorite strict weak ordered set. However, for the purposes of this paper, we will consider subsets of $\mathbb{Z}_{12}$ bijective to musical notes inside an octave. We will also consider strict weak ordered subsets of $\mathbb{Z}_{12}$ as those can be thought of as scales. One of the most basic subsets of $\mathbb{Z}_{12}$ used in creating our music has been $\{0, 2, 4, 5, 7, 9, 11\}$, better known as the major scale. Other scales used can be found in codes in the appendix.

2.1. Spinners of Equal Probabilities.

Definition 2.2. *Let $S : X \to \mathbb{Z}_c$ be a spinner where $X = \{X_0, X_1, ..., X_n\} \sim \mathcal{U}(0, c-1)$.*
*A **spinner of equal probability** can be defined recursively as*

$$S(X_k) = \begin{cases} S(X_0) & = X_0 \quad k = 0 \\ S(X_k) & \equiv (S(X_{k-1}) + X_k) \mod c \quad 0 < k \leq n, \end{cases}$$

in which all $S(X_k)$ are the smallest nonnegative residue, modulo c.

Or defined as partial sums in the following manner:

$$S(X_k) \equiv \left(\sum_{j=0}^{k} X_k \right) \mod c$$

in which all $S(X_k)$ are the smallest nonnegative residue, modulo c.

Although quite unrealistic, the reason for bounding all X_k's between 0 and $c-1$ can be seen much easier on the physical model. It can be thought so that one must spin the wheel less than a complete revolution. However, it is empirical to have this bound when we talk about "rotations" of a spinner.

2.1.1. *Coding*. At its simplest, we would only like to code some spinner whose image has elements from one octave. The initiated variables can be found in the appendix.

```
def majortune(octv,dur):              ## Builds you some sweet tunes

    s = rand.randint(0,len(C)-1)       # pick a random starting note
    T.append(C[s])                     # put note in tune
    for i in range(0,dur-1):           # offset by one because we already
       appended one before the for loop
        r = rand.randint(0,len(C))     # spin the wheel
        r = (s + r)%(len(C))           # new position of spinner
        first = C[r]                   # gives you the note you landed on
        third = C[(r+2)%len(C)]        # gives you the note two indices over (
           also known as its third (but modulated))
        fifth = C[(r+4)%len(C)]        # gives you the note four indices over (
           also known as its fifth (but modulated))
        ranote = C[(p)%len(C)]         # chooses another random note, sometimes
            it's quite good
        T.append(first)                # for single notes
    return T                           # returns your sweet tune
```

To include more octaves, we use the `majoroct(octv)` where the argument it takes is a number that produces the amount of octaves the reader wants greater than 0.

To make chords, we are able to use `T.append(['insert your chord here']).` For example, to make a triad, we would code `T.append([first,third,fifth])` instead of `T.append(first).`

At first one might notice the sound of increasing notes of the spinner. In this case, we can change the range at which the spinner is able to pick indices.

It is important to remember that the method in which we are picking indices is of a discrete uniform distribution, but we can adapt this to be any distribution. For the purposes of the program in the appendix, we only used a discrete uniform distribution to "spin the wheel." However, we can calculate probabilities of different distributions.

At the bottom of the code, the output provided is used to copy-and-paste into a SuperCollider notebook. The use of `Pbind` with arguments `\note`, `\dur`, and `\legato` take values given by sequences of numbers in an array, to which SuperCollider calls `Pseq`.

We would like to remind the reader, that this paper is by no means a SuperCollider tutorial. We use Python to program our code and receive an output, while SuperCollider to give sound to said output.

2.1.2. *Some Analysis.* Some motivation of the following analysis came from manipulating the code for this spinner. It was observed that when changing the parameters of the discrete uniform distribution, the notes that played were sometimes heard in increasing order.

Definition 2.3. *Let $S : \mathbb{Z}_n \to \mathcal{C}$ be a spinner of equal probability. We say $S(X_k)$ is a **rotation** exactly when $S(X_{k-1}) > S(X_k)$.*

Definition 2.4. *Let $R = \{\rho_1, \rho_2, ..., \rho_r\}$ be the set of all rotations of some spinner. Then the arpeggio set* arp*, is notated as* $\text{arp} = \{\alpha_a : \rho_a - \rho_{a-1}, 0 < a \leq r\}$*, where ρ_0 is implied to be exactly zero, but not included in the set.*

These two definitions are not quite intuitive. To give some motivation to what we are trying to prove, one can think of arpeggios in music theory. Rotations are the elements of S that are the bottom note of an arpeggio. The arpeggio set tells us how many notes are in the arpeggio, that is, how many elements are between each rotation.

Problem 2.1. Let $S : X \to \mathbb{Z}_9$ be a spinner of equal probability. The following sequence was produced from S: $\{1, 2, 4, 6, 8, 1, 3, 5, 7, 2, 4, 3, 0, 8, 7\}$. What is the arpeggio set?

Solution 2.1. First, let's take a look at how many rotation there are in this example. We note that $S(5), S(9), S(11), S(12), S(14)$ are all rotations. Because there cannot be any rotations previous to the start of the spinner, $\alpha_1 = 5 - 0 = 5$. The rest follow suit:

$$\begin{aligned}
\alpha_1 &= 5 - 0 = 5 \\
\alpha_2 &= 9 - (5) = 4 \\
\alpha_3 &= 11 - (9) = 2 \\
\alpha_4 &= 12 - (11) = 1 \\
\alpha_5 &= 14 - (12) = 2.
\end{aligned}$$

Thus, the arpeggio set for this spinner, $\text{arp} = \{5, 4, 2, 1, 2\}$

From this example, we are able to see that the elements of the arpeggio set precisely describe the number of elements between two rotations:

$$\{\underbrace{1,2,4,6,8,}_{\text{5 elements}} | \underbrace{1,3,5,7,}_{4} | \underbrace{2,4,}_{2} | \underbrace{\ 3,\ }_{1} | \underbrace{0,8,}_{2} |7\}.$$

Also note that there is no good way to classify the last element of the set, as we can't know exactly what the next element could have been, and thus we can only say it is a rotation.

Theorem 2.1 (Weak Boundedness)**.** Let $S : X \to \mathbb{Z}_c$ be a spinner of equal probability where $X = \{X_0, X_1, ..., X_n\} \sim \mathcal{U}(0, n-1)$. If the arpeggio set, $\text{arp} = \{\alpha : \alpha \text{ is an arpeggio of } S\}$, then $? < \mathbf{E}[\alpha] <?$.

Before starting the proof, we would like to formulate the "Candy Store" problem.

Suppose a customer is in a candy store for the first time in the their life. The candy store has a odd, but strictly enforced rule where all of their customers have to buy candy while blindfolded, but the distribution of this candy is based upon a discrete uniform distribution in volume of candy from 0 to $n-1$ where n is the size of any bag in the candy store.

Once a customer e exactly fills their bag, they must pay for their candy and leave the candy store. Additionally, if a customer overflows their bag by one piece of candy (for example, if a bag holds 10 kilograms, the customer already has 7 kilograms in their bag, and the customer picks a 5 kilogram piece of candy) they must pay for their candy and leave the store).

The catch happens when customers return to the candy store. The next time the customer goes into the store, the owners remember the overflow (if any) from their previous visit and will consider that value in their current visit. In our previous example, the bag will have an overflow of 2 kilograms, and therefore on their next visit, the customer will start with 2 kilograms and pick the remaining candy taking that amount into account.

The question now becomes: on average, how many pieces of candy will a customer get on a trip to the candy store?

It is obvious that the "Candy Store" problem would have an equivalent answer to the $\mathbf{E}[\alpha]$. In fact, it might be easier to study these in a different probabilistic distribution. We suggest a better start would be the use of the Geometric Distribution where $X \sim \text{GEO}(\frac{1}{|\mathcal{C}|})$.

Proof 2.1. ?

2.2. **Spinners of Unequal Probabilities.** A great, but nonetheless obvious, question we can ask after spinners of equal probabilities are spinners of unequal probabilities. Though they have the same underlying structure of spinners, spinners of unequal probabilities are not much different than those of equal probabilities.

Definition 2.5. *We say $a \equiv b \mod 1$ if $a = b - \lfloor b \rfloor$. That is, we would only like for a to be the smallest non-negative decimal residue, i.e. $a \in [0,1)$.*

Definition 2.6. *Let $P = \{P_0, P_1, ..., P_m\}$ be a set of ordered probabilities. with a respective CDF. Also let $X = \{X_0, X_1, ..., X_n\} \sim U(0,1)$ i.i.d. continuous uniform. For functions $S : X \to [0,1)$ where*

$$S(k) \equiv \left(\sum_{i=0}^{k} X_i \right) \mod 1 \quad k \in [0,n]$$

and $T : [0,1) \to P$ such that

$$T(a) = \begin{cases} P_0 & \text{if } a \leq P_0 \\ P_a & \text{if } CDF(P_{a-1}) < a \leq CDF(P_a), \end{cases}$$

*then a **spinner or unequal probability** is the composition, $T \circ S$.*

We do not include the value of 1 in the CDF because of the modulus 1. Nonetheless, $P[X = 0] = P[X = 1] = 0$. Additionally, we tend to truncate the values of the $X_i's$ to a few decimals, as they are much easier to handle for precision.

2.2.1. *Coding*. Coding for spinners of unequal probabilities was not as similar as those of equal probability. It relied heavily on the CDF of the original spinner, and thus a modulus approach was naive and ill-fetched. The code can be found in Appendix A.1.

2.2.2. *Analysis.* We begin by first taking a look at an example of a spinner of unequal probability.

Example 2.1. Let $S : \mathbb{Z}_{100} \to Z_5$ be a spinner with the minor pentatonic scale, $\{0, 3, 5, 7, 10\}$ as its image. The probabilities for each note are given by the following table:

$$\mathbf{P}[\mathbb{Z}_5 = 0] = 0.07168$$
$$\mathbf{P}[\mathbb{Z}_5 = 3] = 0.27432$$
$$\mathbf{P}[\mathbb{Z}_5 = 5] = 0.37143$$
$$\mathbf{P}[\mathbb{Z}_5 = 7] = 0.04052$$
$$\mathbf{P}[\mathbb{Z}_5 = 10] = 0.24205$$

After running the program, the output values of S were given to be:
[3, 0, 10, 10, 3, 10, 3, 10, 5, 3, 5, 3, 5, 10, 5, 3, 10, 10, 10, 5, 10, 3, 3, 3, 3, 10, 10, 10, 5, 5, 5, 5, 5, 5, 3, 5, 5, 0, 7, 0, 7, 3, 0, 3, 5, 3, 10, 5, 5, 10, 10, 5, 5, 5, 10, 0, 10, 5, 0, 3, 10, 10, 3, 10, 5, 7, 3, 3, 5, 5, 10, 3, 5, 3, 5, 5, 5, 10, 3, 5, 3, 10, 5, 5, 3, 3, 5, 3, 3, 5, 5, 5, 5, 10, 5, 5, 5, 0, 3, 3].

Let's count them and see how they match to their true probabilities.

Note	Occurrence	Actual Value	Expected Value	Relative Error
0	7	0.07	0.07168	0.023438
3	28	0.28	0.27431	0.020743
5	38	0.38	0.37143	0.023073
7	3	0.03	0.04052	0.025962
10	24	0.24	0.24205	0.008469

After calculating the relative error, we can see that even at 100 steps into the program, the actual probabilities are close to 2% off from their true probabilities. It is highly likely that if we took a larger data set, the values of the relative error would tend closer to zero.

We can ask the same questions about rotations and arpeggios about spinners of unequal probabilities.

APPENDIX A. CODES

A.1. **A.** Spinners of Equal Probabilities

```
##### Equal Probabilities for all notes!

import numpy
import math
import random as rand
from random import randrange
from random import choice
import os

#C = [0,2,4,5,7,9,11]              ## Major Scale
#C = [0,2,3,5,7,8,10]              ## Minor Scale
#C = [0,2,3,5,7,8,11]              ## Harmonic Minor Scale
#C = [0,2,3,5,7,8,10]              ## Descending Melodic Minor Scale
#C = [0,2,4,7,9]                   ## Major Pentatonic Scale
C = [0,3,5,7,10]                  ## Minor Pentatonic Scale
#C = [0,2,4,6,8,10]                ## Whole Tone Scale
#C = [0,3,5,6,7,10]                ## Blues Scale
#C = [0,4,7]                       ## Triads
CC = [1,2,3,4,5,6,7,8,9,10,11,12]  ## Chromatic Scale

T = [] ## Tune
TT = []## Tune

R = [] ## Some extra stuff
CR = []## Some extra stuff
P = [] ## Some extra stuff

LE = [] ## Length of the chord

## Builds octaves in a major scale
def majoroct(octv):
    if octv > 1:
        for i in range(1,octv):
            for j in range(0,len(C)):
                c = C[j]+12
                C.append(c)
    NC = C
    return NC

## Builds octaves in a chromatic scale
def chromoct(octv):
    for i in range(1,octv):
```

```
        for j in range(0,len(C)):
            cc = CC[j]+12
            CC.append(cc)
    NCC = CC
    return NCC

def majortune(octv,dur):                ## Builds you some sweet tunes
    majoroct(octv)
    s = rand.randint(0,len(C))          # pick a random starting note
    for i in range(0,dur):              # offset by one because we already
        appended one before the for loop
        r = rand.randint(1,len(C)-1)            # spin the wheel ## You can
            change this to whatever you want!
        s = ((s + r)%(len(C)))          # new position of spinner
        first = C[s]                    # gives you the note you landed on
        third = C[(s+2)%len(C)]         # gives you the note two indices over
             (also known as its third (but modulated))
        fifth = C[(s+4)%len(C)]         # gives you the note four indices
            over (also known as its fifth (but modulated))
        p = rand.randint(0,len(C)-1)    # picks a random note
        ranote = C[p]                   # chord with a random note, sometimes
             it's quite good
#        T.append([first,fifth,ranote])  ## for excruciating pain
#        T.append([first,third,fifth])   ## for tri-chords
        T.append([first,fifth])         ## for Power Chords
#        T.append([first,third])         ## for 1 and 3
#        T.append([first,ranote])        ## for mild pain
#        T.append(first)                 ## for single notes
    return T                            # returns your sweet tunes

def chromtune(octv,dur):                ## Builds you some sweet tunes
    chromoct(octv)
    s = rand.randint(0,len(CC))         # pick a random starting note
    for i in range(0,dur):              # offset by one because we already
        appended one before the for loop
        r = rand.randint(1,8)           # spin the wheel ## You can change
            this to whatever you want!
        s = ((s + r)%(len(CC)))         # new position of spinner
        first = CC[s]                   # gives you the note you landed on
        third = CC[(s+4)%len(CC)]       # gives you the note two indices over
             (also known as its third (but modulated))
        fifth = CC[(s+7)%len(CC)]       # gives you the note four indices
            over (also known as its fifth (but modulated))
        p = rand.randint(0,len(CC))     # picks a random note
        ranote = CC[(p)%len(CC)]        # chord with a random note, sometimes
             it's quite good
```

```
#         T.append([first,fifth,ranote])  ## for excruciating pain
#         T.append([first,third,fifth])   ## for tri-chords
#         T.append([first,fifth])         ## for Power Chords
#         T.append([first,ranote])        ## for mild pain
        T.append(first)                  ## for single notes
    return T                              # returns your sweet tunes
def length():
    n = numpy.rand.geometric()
    return LE

## Durations of some form that isn't too complicated
def length2(octv,dur):
    for i in range(0,dur):
        if i%2 == 0:
            a = .33
            LE.append(a)
        else:
            a = 0.11
            LE.append(a)
    return LE

## Durations from a set of data:
def length3(octv,dur):
    B = [0.25, 0.5, 0.75, 1, 1.25, 1.5, 2]
    for i in range(0,dur-1):
        b = rand.choice(B)
        B.append(b)
    B.append(2)
    return B

def durationer(a,b):
    r = rand.uniform(a,b)
    r = round(r,2)
    return r

def offset(a,b):
    r = rand.randint(a,b)
    return r

print "(\nPbind(\n\\note,␣Pseq(\n␣\n\t", majortune(3,100), "+", offset(-3,3)
print "\n\n\t,0.75),\n\\dur,␣Pseq(\n\n", length(3,100),
print "\n\n\t,1),\n\\legato,",durationer(1.5,2.75),",\n).play;\n)"
```

A.2. **B.** Spinners of Unequal Probabilities

```
###### Random Probabilities for Different Notes
```

```
import math
import numpy
import random as rand

#C = [0,2,4,5,7,9,11]              ## Major Scale
#C = [0,2,3,5,7,8,10]              ## Minor Scale
#C = [0,2,3,5,7,8,11]              ## Harmonic Minor Scale
#C = [0,2,3,5,7,8,10]              ## Descending Melodic Minor Scale
C = [0,2,4,7,9]                   ## Major Pentatonic Scale
#C = [0,3,5,7,10]                  ## Minor Pentatonic Scale
#C = [0,2,4,6,8,10]                ## Whole Tone Scale
#C = [0,3,5,6,7,10]                ## Blues Scale
#C = [0,4,7]                       ## Triads

P = [0]  # Takes the probabilities of the colors
PP = []
CDF = [] # the cummulative distribution function
CDFCDF = []
T = [] # the tune
TT = []
LE = []
t_third = []
t_fifth = []
t_random = []
def prob(octv, a,b, dur): # Arguments: beta distribution(alpha, beta),
    duration of song
    if octv > 1:
        for i in range(1,octv):
            for j in range(0,len(C)):
                c = C[j]+12
                C.append(c)
    for i in range(0, len(C)):
        p = rand.betavariate(a,b) # Picks random numbers based on the beta
            distribution
        P.append(p) # Puts the value into the probanility array
    summ = sum(P) # Adds up all of the values in the probability array
    for i in range(0, len(C)+1):
        P[i] = round(1.0*P[i]/summ,5) # Divides all of the original random
            numbers by the sum to get probabilities
    summm = sum(P) # adjusts the new cdf which should equal 1
    cdf = 0
    for i in range(0, len(C)+1):
        cdf = round(cdf + P[i],5)
        CDF.append(cdf) #gives you the CDF array
    P.remove(0) # takes out zero as a probability
```

```
    r = round(rand.random(),6) # This is the starting random note
    for j in range(0, dur):
        for i in range(0, len(C)):
            if CDF[i] < r and r < CDF[i+1]: # looks if the random number is
                between two CDF array elements
                 t = C[i] # returns a note
                 t_third = C[(i+2)%len(C)]
                 t_fifth = C[(i+4)%len(C)]
#                 T.append(t)   # Single Note
                 T.append([t,t_third]) #First and Third
#                 T.append([t,t_fifth]) # Power Chord
#                 T.append([t,t_third,t_fifth]) # Triad
        s = round(rand.random(),6) # This is the spinner part
        r = (r+s) - (r+s)//1 # The circle modulates
    return T, CDF, P
# Arguments: beta distribution(alpha, beta), duration of song
def length(dur):
    for i in range(0, dur):
        leng = rand.randint(1, 1792)
        leng = leng%100
        if leng == 0:
            leng + 1
        leng = round(1- leng/100.0,2)
        LE.append(leng)
    return LE

def legato(a,b):
    r = rand.uniform(a,b)
    r = round(r,2)
    return r

def offset(a,b):
    r = rand.randint(a,b)
    return r

def spinner(octv,a,b,dur):
    prob(octv,a,b,dur)
    TT = T
    PP = P
    print PP
    return TT

spin = spinner(1,3,6,100)
print "(\nPbind(\n\\note,␣Pseq(\n␣\n\t", spin, "+", offset(-3,3)
print "\n\n\t,1),\n\\dur,␣Pseq(\n\n", length(100),
print "\n\n\t,1),\n\\legato,",legato(1.5,2.75),",\n).play;\n)"
```

APPENDIX B. EXAMPLES

B.1. **A.** Spinners of Equal Probabilities

```
/*
Triadness
Natural Major Scale, Four Octaves, Seven Offset
Chords built as triad based on bottom note
Duration of chord based on length of scale
Legato>2
*/
(
Pbind(
\note, Pseq(
   [[5, 13, 20], [41, 1, 8], [5, 13, 20], [20, 29, 13], [8, 17, 13], [32, 41,
      1], [17, 25, 32], [32, 17, 25], [29, 37, 44], [5, 13, 20], [25, 32,
      29], [13, 20, 29], [32, 41, 1], [1, 8, 17], [1, 8, 17], [41, 1, 8],
      [25, 32, 29], [17, 13, 20], [1, 8, 17], [25, 32, 41], [25, 32, 17], [1,
       8, 17], [32, 41, 1], [13, 20, 29], [32, 41, 1], [32, 17, 25], [32, 41,
       1], [17, 25, 32], [1, 8, 17], [20, 29, 25], [25, 32, 29], [8, 17, 13],
       [13, 20, 29], [17, 25, 32], [8, 17, 13], [1, 8, 17], [29, 13, 20],
      [13, 20, 29], [20, 17, 25], [41, 1, 8], [13, 20, 29], [29, 13, 20],
      [37, 44, 5], [1, 8, 17], [17, 13, 20], [25, 32, 29], [44, 5, 13], [29,
      13, 20], [32, 41, 1], [13, 20, 17], [13, 20, 29], [13, 20, 17], [13,
      20, 29], [20, 29, 25], [17, 25, 32], [13, 20, 29], [20, 29, 25], [20,
      29, 13], [25, 32, 17], [13, 20, 29], [32, 29, 37], [25, 32, 41], [1, 8,
       17], [20, 29, 25], [17, 25, 32], [20, 17, 25], [32, 17, 25], [8, 17,
      13], [32, 41, 1], [8, 17, 13], [41, 1, 8], [20, 29, 13], [8, 17, 13],
      [25, 32, 17], [20, 29, 25], [32, 41, 1], [13, 20, 29], [32, 17, 25],
      [13, 20, 29], [17, 25, 32], [17, 13, 20], [20, 17, 25], [20, 29, 13],
      [25, 32, 17], [13, 20, 29], [13, 20, 29], [17, 13, 20], [8, 17, 13],
      [29, 37, 44], [1, 8, 17], [13, 20, 17], [29, 13, 20], [32, 29, 37],
      [20, 29, 25], [13, 20, 29], [13, 20, 29], [13, 20, 29], [41, 1, 8],
      [44, 5, 13], [1, 5, 8]] + -7
       ,1),
\dur, Pseq(
   [0.74, 0.43, 0.69, 0.2, 0.67, 0.31, 0.68, 0.17, 0.14, 0.07, 0.23, 0.7,
      0.66, 0.94, 0.12, 0.86, 0.42, 0.89, 0.13, 0.81, 0.36, 0.57, 0.7, 0.47,
      0.72, 0.83, 0.83, 0.89, 0.46, 0.07, 0.17, 0.53, 0.04, 0.77, 0.56, 0.58,
       0.64, 0.49, 0.76, 0.5, 0.23, 0.62, 0.08, 0.11, 0.33, 0.89, 0.21, 0.76,
       0.8, 0.42, 0.67, 0.76, 0.99, 0.55, 0.93, 0.27, 0.36, 0.28, 0.42, 0.07,
       0.99, 0.61, 0.19, 0.44, 0.04, 0.83, 0.44, 0.35, 0.34, 0.82, 0.5, 0.08,
       0.29, 0.7, 0.16, 0.3, 0.2, 0.75, 0.41, 0.99, 0.47, 0.65, 0.19, 0.8,
      0.73, 0.66, 0.49, 0.51, 0.56, 0.78, 0.52, 0.54, 0.96, 0.6, 0.29, 0.65,
      0.44, 0.17, 0.32, 0.95]
       ,1),
```

```
\legato, 2.34 ,
).play;
)


/*
Sadder song, minor
Natural Minor Scale, One Octave, No offset
One note at a time
Predetermined Duration Sizes
Legato>1
*/
(
Pbind(
\note, Pseq(

        [3, 11, 8, 9, 1, 3, 4, 6, 9, 11, 3, 4, 1, 1, 6, 1, 1, 9, 6, 11, 9, 11,
            3, 11, 4, 9, 1, 3, 3, 6, 9, 3, 4, 6, 6, 6, 3, 6, 6, 4, 3, 1, 9,
           4, 3, 9, 3, 1, 1, 3, 8, 1, 9, 8, 8, 1, 6, 6, 6, 9, 11, 4, 6, 11,
           1, 8, 8, 4, 4, 1, 11, 11, 6, 4, 4, 8, 3, 3, 4, 9, 4, 1, 9, 8, 4,
           4, 9, 9, 3, 1, 1, 4, 11, 11, 11, 3, 4, 3, 4, 9, 4, 1]-1

        ,5),
\dur, Pseq(

[2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5,
   0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1,
    0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5,
    2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1,
   0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5,
   2, 1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2, 1]

        ,5),
\legato,2,
).play;
)
```

B.2. **B.** Spinners of Unequal Probabilities

```
/*
Probabilistic Simple Tune
Uses the 1, 5, 6, 8, 13 of a chromatic scale, One Octave, Seven Offset
PDF : [1 : 0.30563, 5 : 0.2129, 6 : 0.21674, 8 : 0.17189, 13 : 0.09284]
Random durations
Legato > 2
*/
```

```
(
Pbind(
\note, Pseq(

        [5, 8, 1, 5, 1, 8, 5, 1, 1, 6, 1, 6, 6, 13, 8, 1, 6, 6, 6, 5, 5, 5, 6,
            5, 6, 1, 1, 8, 13, 1, 1, 5, 5, 5, 1, 6, 6, 8, 8, 5, 1, 6, 8, 8,
           1, 1, 1, 5, 8, 1] + -7
        ,1),
\dur, Pseq(
[0.89, 0.91, 0.91, 0.91, 0.86, 0.93, 0.92, 0.84, 0.87, 0.94, 0.93, 0.86, 0.95,
    0.87, 0.97, 0.94, 0.92, 0.85, 0.83, 0.94, 0.97, 0.89, 0.83, 0.9, 0.82,
   0.9, 0.96, 0.87, 0.88, 0.89, 0.98, 0.92, 0.92, 0.93, 0.87, 0.91, 0.85,
   0.93, 0.87, 0.93, 0.93, 0.83, 0.9, 0.88, 0.88, 0.91, 0.84, 0.94, 0.9,
   1.3]*0.3
        ,1),
\legato, 2.27 ,
).play;
)

/*
Probabilistic Triad Tune
Uses the major scale, One Octave, Three Offset
PDF : [1 : 0.16643, 3 : 0.21488, 5 : 0.08487, 6 : 0.14073, 8 : 0.26036, 10 :
   0.06221, 12 : 0.07051]
Random durations
Legato > 1
*/

(
Pbind(
\note, Pseq(
```

```
        [[1, 5, 8], [8, 12, 3], [1, 5, 8], [8, 12, 3], [1, 5, 8], [3, 6, 10],
           [12, 3, 6], [6, 10, 1], [1, 5, 8], [1, 5, 8], [3, 6, 10], [3, 6,
           10], [8, 12, 3], [1, 5, 8], [3, 6, 10], [6, 10, 1], [1, 5, 8], [8,
            12, 3], [6, 10, 1], [1, 5, 8], [1, 5, 8], [3, 6, 10], [6, 10, 1],
            [5, 8, 12], [3, 6, 10], [8, 12, 3], [3, 6, 10], [12, 3, 6], [8,
           12, 3], [8, 12, 3], [3, 6, 10], [3, 6, 10], [8, 12, 3], [3, 6,
           10], [8, 12, 3], [3, 6, 10], [6, 10, 1], [8, 12, 3], [3, 6, 10],
           [6, 10, 1], [3, 6, 10], [5, 8, 12], [8, 12, 3], [6, 10, 1], [3, 6,
            10], [3, 6, 10], [5, 8, 12], [1, 5, 8], [1, 5, 8], [10, 1, 5],
           [1, 5, 8], [6, 10, 1], [3, 6, 10], [6, 10, 1], [5, 8, 12], [1, 5,
           8], [1, 5, 8], [3, 6, 10], [6, 10, 1], [10, 1, 5], [8, 12, 3], [8,
            12, 3], [8, 12, 3], [8, 12, 3], [6, 10, 1], [1, 5, 8], [8, 12,
           3], [3, 6, 10], [12, 3, 6], [3, 6, 10], [1, 5, 8], [8, 12, 3],
           [12, 3, 6], [3, 6, 10], [3, 6, 10], [1, 5, 8], [8, 12, 3], [8, 12,
            3], [6, 10, 1], [8, 12, 3], [8, 12, 3], [6, 10, 1], [1, 5, 8],
           [6, 10, 1], [8, 12, 3], [8, 12, 3], [1, 5, 8], [1, 5, 8], [6, 10,
           1], [6, 10, 1], [8, 12, 3], [3, 6, 10], [3, 6, 10], [5, 8, 12],
           [1, 5, 8], [3, 6, 10], [1, 5, 8], [1, 5, 8], [3, 6, 10], [6, 10,
           1]] + -3


        ,1),
\dur, Pseq(

[0.46, 0.77, 0.41, 0.48, 0.58, 0.96, 0.65, 0.29, 0.02, 0.02, 0.85, 0.32, 0.34,
    0.77, 0.3, 0.63, 0.73, 0.78, 0.57, 0.37, 0.64, 0.78, 0.11, 0.67, 0.08,
   0.58, 0.22, 0.08, 0.21, 0.55, 0.96, 0.73, 0.8, 0.27, 0.56, 0.8, 0.25,
   0.34, 0.04, 0.47, 0.34, 0.91, 0.94, 0.22, 0.06, 0.73, 0.46, 0.82, 0.78,
   0.64, 0.48, 0.15, 0.69, 0.23, 0.06, 0.58, 0.47, 0.85, 0.94, 0.64, 0.94,
   0.31, 0.9, 0.55, 0.62, 0.6, 0.86, 0.17, 0.54, 0.66, 0.1, 0.42, 0.02, 0.41,
    0.67, 0.77, 0.52, 0.5, 0.17, 0.2, 0.58, 0.26, 0.09, 0.28, 0.12, 0.56,
   0.71, 0.8, 0.08, 0.93, 0.59, 0.4, 0.5, 0.46, 0.46, 0.58, 0.16, 0.04, 0.3,
   0.56]

        ,1),
\legato, 1.2 ,
).play;
)
```