Chapter 3. Meeting 3, Approaches: Distributions and Stochastics

3.1. Announcements

- Download: most recent athenaCL

  http://code.google.com/p/athenacl

3.2. Reading: Ames: A Catalog of Statistical Distributions

- Ames, C. 1991. “A Catalog of Statistical Distributions: Techniques for Transforming Random, Determinate and Chaotic Sequences.” *Leonardo Music Journal* 1(1): 55-70.
- What does Ames mean by balance, and that there can be a balance that is not fair?
- What is meant by a weight? Why is this term preferable to alternatives?
- The use of statistics here might be considered outside of the discipline of statistics: why?
- Which musical parameters are better suited for discrete values? Which for continuous values?
- Are any distributions dependent on past occurrences?
- Why might the Law of Large Numbers make working with distributions difficult in musical contexts?
- In terms of the distribution output, what are time domain and frequency domain graphs?
- What is the relationship between the Poisson distribution and the Exponential distribution?
- Ames notes that, when working with some distributions, values may have to be discarded: why? What does this say about working with distributions?

3.3. ParameterObjects

- Reusable value selectors and generators

- Can be created and controlled with strings of comma-separated lists
- Values in ParameterObjects can be strings (without quotes), numbers, or lists (delimited by parenthesis or brackets)
- In some cases ParameterObjects, enclosed as a list, can be used inside of other ParameterObjects to generate values
- Three types of ParameterObjects
  - Generator: produce values based on arguments alone
  - Rhythm: specialized for rhythm creation
  - Filter: specialized for transforming values produced from a Texture
- Complete documentation for ParameterObjects, and samples, can be found here:

  http://www.flexatone.net/athenaDocs/www/ax03.htm
- ParameterObject names and string values can always be provided with acronyms
- Trailing arguments, when not provided, are automatically supplied

3.4. ParameterObjects: Viewing Arguments and Output

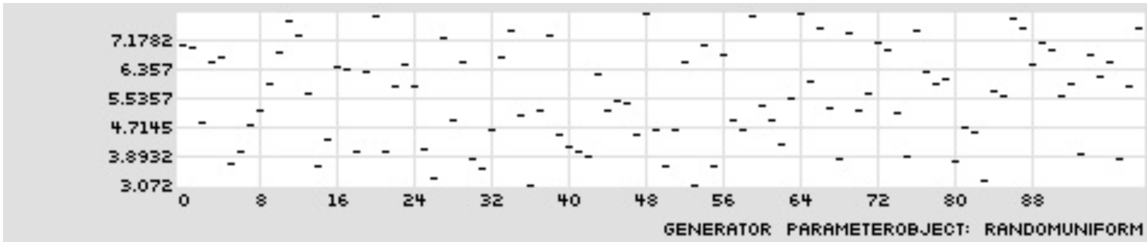
- TPls: view a list of all available ParameterObjects
- TPv: vie detailed documentation for one or more ParameterObjects

```
pi{}ti{} :: tpv ru
Generator ParameterObject
{name,documentation}
RandomUniform       randomUniform, min, max
                    Description: Provides random numbers between 0 and 1 within an
uniform distribution.
                    This value is scaled within the range designated by min and max;
min and max may be
                    specified with ParameterObjects. Note: values are evenly
distributed between min and
                    max. Arguments: (1) name, (2) min, (3) max
```

- TPmap: create a graphical output providing a number of values and a ParameterObject name

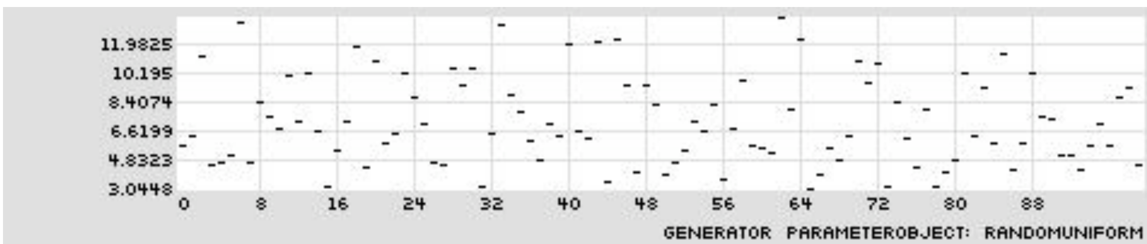
  Note that, when providing arguments from the command-line, spaces cannot be used between ParameterObject arguments

```
pi{}ti{} :: tpmap 100 ru,3,8
randomUniform, (constant, 3), (constant, 8)
TPmap display complete.
```



- With a nested ParameterObject for the maximum value

```
pi{}ti{} :: tpmap 100 ru,3,(ru,8,15)
randomUniform, (constant, 3), (randomUniform, (constant, 8), (constant, 15))
TPmap display complete.
```



3.5. Configuring Graphical Outputs in athenaCL

- athenaCL supports numerous types of graphical outputs, some with various dependencies
- Output formats:
  - JPG, PNG: requires working installation of the Python Imaging Library (PIL)

    Windows: http://www.pythonware.com/products/pil

    Others: not so easy for Python 2.6 (easier for Python 2.5)
  - TK: uses the TK GUI system that ships with Python

    Works for full installs of Python 2.6 on Windows, Mac, Others
  - EPS: works on all Pythons on all platforms
- APgfx: set graphical output preferences

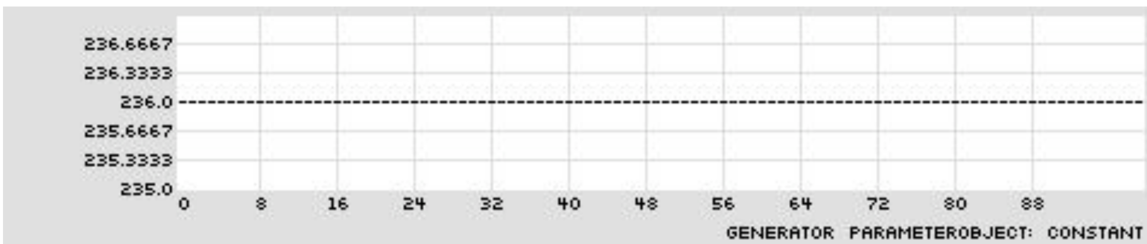
```
pi{}ti{} :: apgfx
active graphics format: png.
select text, eps, tk, jpg, png. (t, e, k, j, or p): p
graphics format changed to png.
```

- Use APea to set the imageViewer and psViewer applications if not already set properly

3.6. The Constant ParameterObject

- The most simple ParameterObject

```
pi{}ti{} :: tpv constant
Generator ParameterObject
{name,documentation}
Constant             constant, value
                     Description: Return a constant string or numeric value.
Arguments: (1) name, (2)
                     value
```



3.7. Continuous and Discrete Stochastic Distributions as ParameterObjects

- Discrete
  - BasketGen
  - Continuous POs put through the Quantize PO or other POs
- Continuous
  - RandomUniform
  - RandomGauss
  - RandomBeta
  - RandomExponential and RandomInverseExponential
  - Many others...

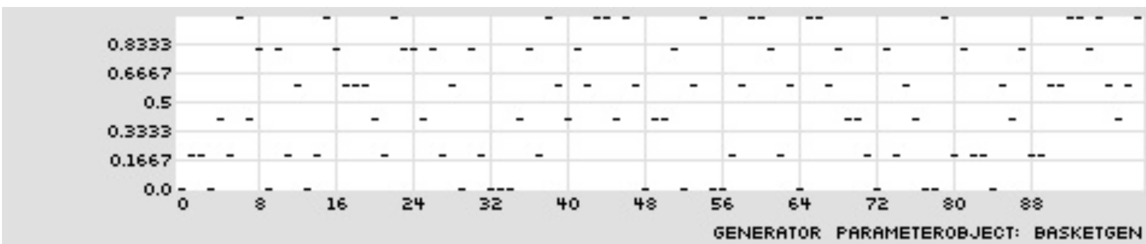
3.8. Discrete Stochastic Distributions as ParameterObjects

- BasketGen: the ball and urn (or basket) paradigm
- Documentation with TPv

```
:: tpv bg
Generator ParameterObject
{name,documentation}
BasketGen           basketGen, selectionString, valueList
                    Description: Chooses values from a user-supplied list
                    (valueList). Values can be strings or numbers. Values are
                    chosen from this list using the selector specified by the
                    selectionString argument. Arguments: (1) name, (2)
                    selectionString {'randomChoice', 'randomWalk',
                    'randomPermutate', 'orderedCyclic',
                    'orderedCyclicRetrograde', 'orderedOscillate'}, (3)
                    valueList
```

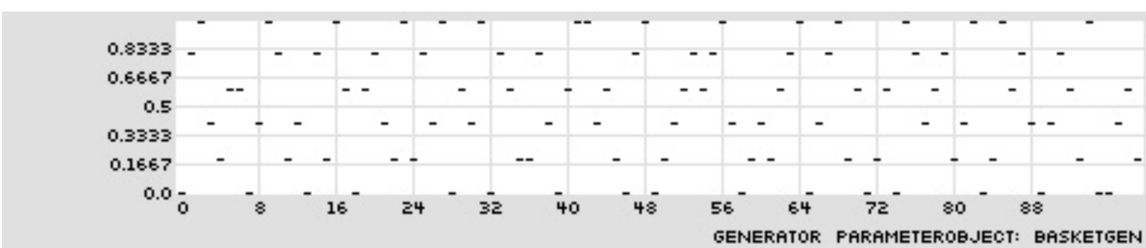
- Selection methods
  - randomChoice: random selection with replacement

```
pi{}ti{} :: tpmap 100 bg,rc,(0,.2,.4,.6,.8,1)
basketGen, randomChoice, (0,0.2,0.4,0.6,0.8,1)
TPmap display complete.
```



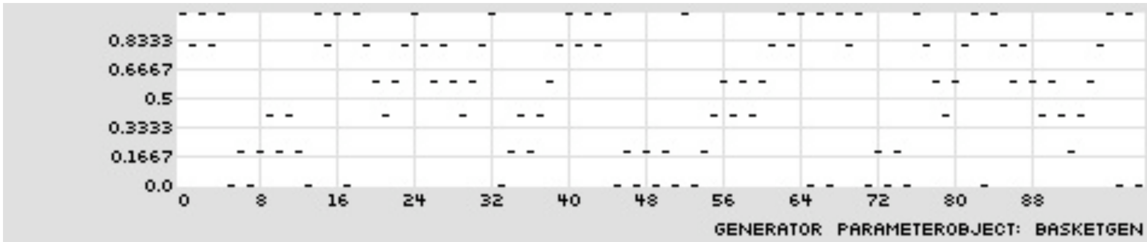
  - randomPermutate: random selection without replacement

```
pi{}ti{} :: tpmap 100 bg,rp,(0,.2,.4,.6,.8,1)
basketGen, randomPermutate, (0,0.2,0.4,0.6,0.8,1)
TPmap display complete.
```



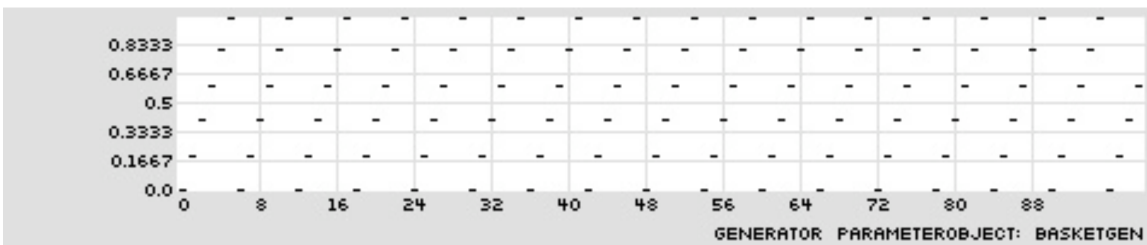
  - randomWalk: random up/down movement along order of list, with wrapping

```
pi{}ti{} :: tpmap 100 bg,rw,(0,.2,.4,.6,.8,1)
basketGen, randomChoice, (0,0.2,0.4,0.6,0.8,1)
TPmap display complete.
```



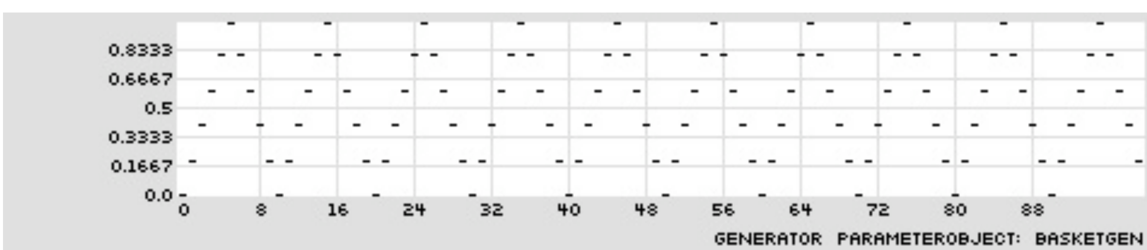
- orderedCyclic: looping

```
pi{}ti{} :: tpmap 100 bg,oc,(0,.2,.4,.6,.8,1)
basketGen, orderedCyclic, (0,0.2,0.4,0.6,0.8,1)
TPmap display complete.
```



- orderedOscillate: oscillating

```
pi{}ti{} :: tpmap 100 bg,oo,(0,.2,.4,.6,.8,1)
basketGen, orderedOscillate, (0,0.2,0.4,0.6,0.8,1)
TPmap display complete.
```



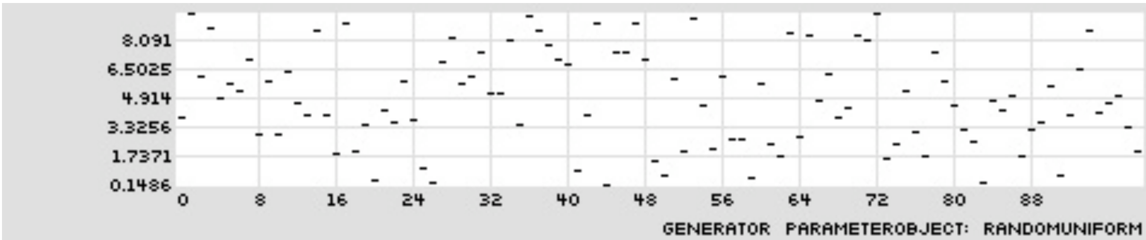
- By configuring the values drawn from, discrete uniform, Bernoulli, and binomial distributions can be modeled

3.9. Continuous Stochastic Distributions as ParameterObjects

- RandomUniform: continuous uniform distribution

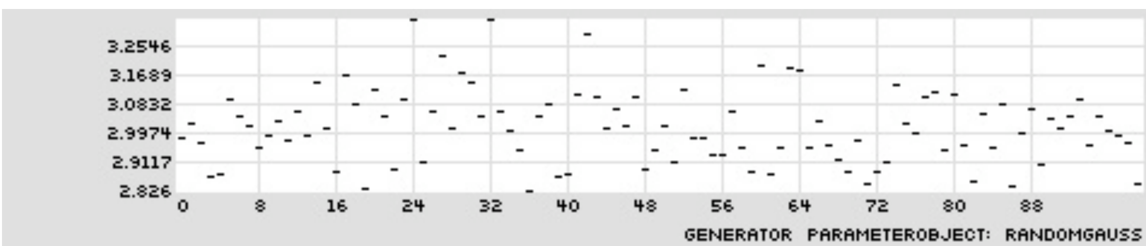
  scaled between 0 and 10

```
pi{}ti{} :: tpmap 100 ru,0,10
randomUniform, (constant, 0), (constant, 10)
TPmap display complete.
```



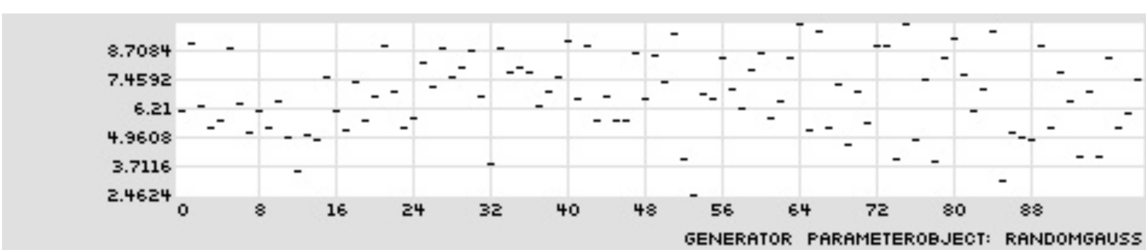
- RandomGauss: normal distribution, arguments mu and sigma
  - mu: center of distribution, between 0 and 1
  - sigma: deviation around center, where .001 is little deviation
  - mu at .3, sigma at .01, scaled between 0 and 10

```
pi{}ti{} :: tpmap 100 rg,.3,.01,0,10
randomGauss, 0.3, 0.01, (constant, 0), (constant, 10)
TPmap display complete.
```



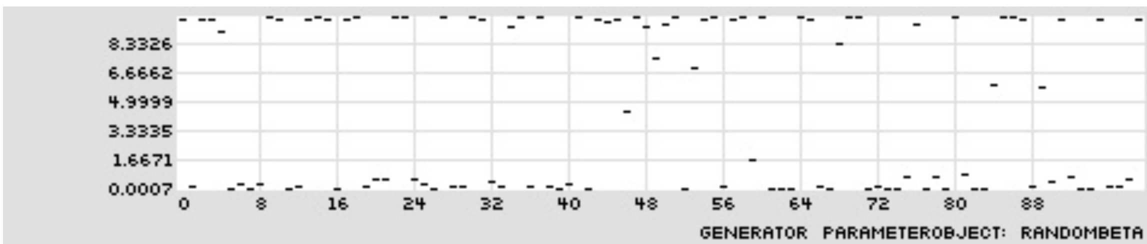
  - mu at .7, sigma at .2, scaled between 0 and 10

```
pi{}ti{} :: tpmap 100 rg,.7,.2,0,10
randomGauss, 0.7, 0.2, (constant, 0), (constant, 10)
TPmap display complete.
```



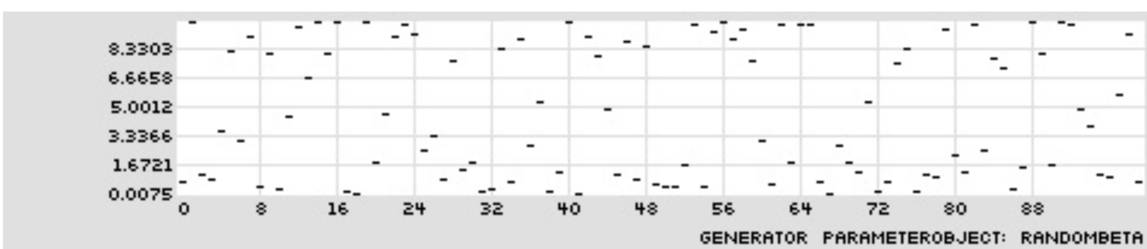
- RandomBeta: arguments alpha and beta
    - This implementation is different than Ames (1991)
    - alpha and beta: low values increase draw to boundaries
    - alpha and beta: large values approach a uniform distribution
    - alpha at .1, beta at .1, scaled between 0 and 10

```
pi{}ti{} :: tpmap 100 rb,0.1,0.1,0,10
randomBeta, 0.1, 0.1, (constant, 0), (constant, 10)
TPmap display complete.
```



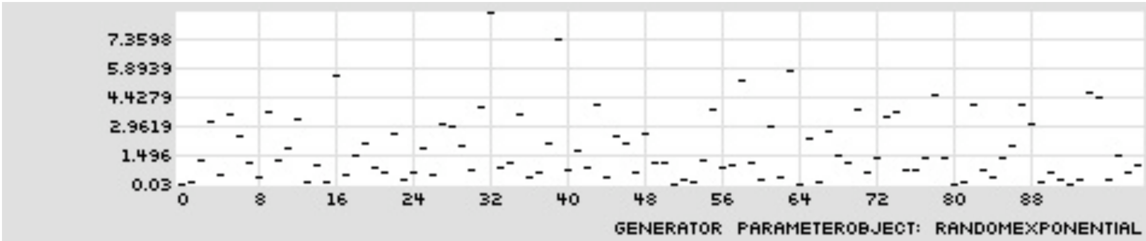
- alpha at .3, beta at .3, scaled between 0 and 10

```
pi{}ti{} :: tpmap 100 rb,.3,.3,0,10
randomBeta, 0.3, 0.3, (constant, 0), (constant, 10)
TPmap display complete.
```



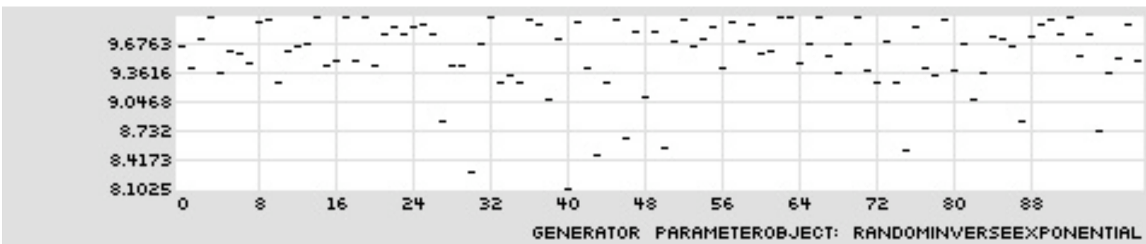
- RandomExponential and RandomInverseExponential
    - lambda: larger values create a tighter pull to to one boundary
    - exponential, lambda at 5, scaled between 0 and 10

```
pi{}ti{} :: tpmap 100 re,5,0,10
randomExponential, 5.0, (constant, 0), (constant, 10)
TPmap display complete.
```



- inverse exponential, lambda at 20, scaled between 0 and 10

```
pi{}ti{} :: tpmap 100 rie,20,0,10
randomInverseExponential, 20.0, (constant, 0), (constant, 10)
TPmap display complete.
```



- For all generators min and max can be embedded POs

3.10. Working with athenaCL

- Often best to use interactive mode for testing values, quick sketches, setting preferences
- Best to use a Python script for composing or other work
- Same preferences used in interactive mode are used in scripts
- For examples, the presence of the command prompt designates that athenaCL is in interactive mode

```
pi{}ti{} ::
```

3.11. Configuring Amplitudes

- Amplitudes in athenaCL are represented within the unit interval (0, 1)
- After creating texture, we can edit the amplitude with the TIe command
- The TIe command needs an argument for what Texture parameter to edit: enter “a” is for amplitude

- Parameter abbreviations can be found with the TIv command

- Setting the amplitude to a RandomUniform value between 0 and 1 [03a.py]

```
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()

ath.cmd('emo mp')

# create a new texture with instrument 45
ath.cmd('tin a 45')

# edit the amplitude of the texture to be RandomUniform between .1 and 1
ath.cmd('tie a ru,.1,1')
ath.cmd('eln')
ath.cmd('elh')
```

- Two parts, one with RandomUniform amplitudes, another with RandomExponential [03b.py]

  Note that textures have to have different names

```
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()

ath.cmd('emo mp')

# create a new texture with instrument 45
ath.cmd('tin a 45')
ath.cmd('tie a ru,.1,1')

# create a new texture with instrument 65
# texture must have a different name
ath.cmd('tin b 65')
ath.cmd('tie a re,15,.2,1')

ath.cmd('eln')
ath.cmd('elh')
```

- Three parts, RandomUniform, RandomExponential, and RandomBeta amplitudes [03c.py]

```
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()

ath.cmd('emo mp')

# create a new texture with instrument 45
ath.cmd('tin a 45')
ath.cmd('tie a ru,.1,1')

# create a new texture with instrument 65
ath.cmd('tin b 65')
ath.cmd('tie a re,15,.2,1')

# create a new texture with instrument 53
ath.cmd('tin c 53')
ath.cmd('tie a rb,.1,.1,.3,.7')

ath.cmd('eln')
ath.cmd('elh')
```

3.12. Duration and Sustain

- Duration
  - The temporal space of an event
  - If events are packed end to end, the time of the next event
  - If a notated event, the written rhythm
- Sustain
  - The sounding (actual) time of the event
  - A scalar applied to the duration
  - A scalar of 0.2 would suggest a staccato (shortened) event
  - A scalar of 1.2 would create overlapping events

3.13. The Pulse Triple

- athenaCL supports both absolute and relative rhythm values
- The PulseTriple is relative to the beat-defining tempo and made of three values
  - Divisor: divides the tempo beat duration
  - Multiplier: scales the value divided
  - Accent: a rhythm-specific amplitude value, between 0 (o) and 1 (+) (or with symbolic dynamics: mp, mf, etc)
- Conventional rhythms can be easily expressed
  - (4,1,1): 1/4th of a beat (if the beat is a quarter, a sixteenth note)
  - (4,3,1): 3/4ths of a beat (if the beat is a quarter, a dotted eighth note)
  - (1,4,1): 4 beats (if the beat is a quarter, a whole note)
  - (3,1,1): 1/3rd of a beat (if the beat is a quarter, a triplet eighth)
  - (5,8,1): 8/5ths of a beat
- Representational redundancy may be useful
  - (4,2,1) is the same as (2,1,1)

- (1,5,1) is the same as (4,20,1)

3.14. Basic Rhythm ParameterObjects

- PulseTriple: create PulseTriples from embedded ParameterObjects

```
pi{}ti{} :: tpv pulsetriple
Rhythm Generator ParameterObject
{name,documentation}
PulseTriple         pulseTriple, parameterObject, parameterObject, parameterObject,
parameterObject
                    Description: Produces Pulse sequences with four Generator
ParameterObjects that
                    directly specify Pulse triple values and a sustain scalar. The
Generators specify
                    Pulse divisor, multiplier, accent, and sustain scalar. Floating-
point divisor and
                    multiplier values are treated as probabilistic weightings. Note:
divisor and
                    multiplier values of 0 are not permitted and are replaced by 1;
the absolute value
                    is taken of all values. Arguments: (1) name, (2) parameterObject
{pulse divisor},
                    (3) parameterObject {pulse multiplier}, (4) parameterObject
{accent value between 0
                    and 1}, (5) parameterObject {sustain scalar greater than 0}
```

- ConvertSecond: create durations form values in seconds

```
pi{}ti{} :: tpv cs
Rhythm Generator ParameterObject
{name,documentation}
ConvertSecond       convertSecond, parameterObject
                    Description: Allows the use of a Generator ParameterObject to
create rhythm
                    durations. Values from this ParameterObject are interpreted as
equal Pulse duration
                    and sustain values in seconds. Accent values are fixed at 1.
Note: when using this
                    Rhythm Generator, tempo information (bpm) has no effect on event
timing. Arguments:
                    (1) name, (2) parameterObject {duration values in seconds}
```

3.15. Configuring Rhythms

- After creating texture, we can edit the rhythm with the TIe command
- The TIe command needs an argument for what Texture parameter to edit: enter “r” for rhythm
- Using basketGen to control the multiplier [03d.py]

```
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()

ath.cmd(“emo mp”)

ath.cmd(“tin a 45”)
```

```
ath.cmd("tie a rb,.3,.3,.5,.8")
ath.cmd("tie r pt,(c,4),(bg,oc,(3,3,2)),(c,1)")

ath.cmd("eln")
ath.cmd("elh")
```

- Using two basketGens to control multiplier and divisor independently [03e.py]

```
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()

ath.cmd("emo mp")

ath.cmd("tin a 45")
ath.cmd("tie a rb,.3,.3,.4,.8")
ath.cmd("tie r pt,(c,4),(bg,oc,(3,3,2)),(c,1)")

ath.cmd("tin b 65")
ath.cmd("tie a re,15,.3,1")
ath.cmd("tie r pt,(bg,rp,(2,1,1,1)),(c,1),(c,1)")

ath.cmd("eln")
ath.cmd("elh")
```

- Using two basketGens to control multiplier and divisor independently [03f.py]

```
from athenaCL.libATH import athenaObj
ath = athenaObj.Interpreter()

ath.cmd("emo mp")

ath.cmd("tin a 45")
ath.cmd("tie a rb,.3,.3,.4,.8")
ath.cmd("tie r pt,(c,4),(bg,oc,(3,3,2)),(c,1)")

ath.cmd("tin b 65")
ath.cmd("tie a re,15,.3,1")
ath.cmd("tie r pt,(bg,rp,(2,1,1,1)),(c,1),(c,1)")

ath.cmd("tin c 67")
ath.cmd("tie a rb,.1,.1,.4,.6")
ath.cmd("tie r cs,(rb,.2,.2,.01,1.5)")

ath.cmd("eln")
ath.cmd("elh")
```

3.16. Configuring Time Range

- After creating texture, we can edit the time range with the TIe command
- The TIe command needs an argument for what Texture parameter to edit: enter "t" for time range
- Enter two values in seconds separated by a comma
- Staggering the entrances of three parts [03g.py]

```
from athenaCL.libATH import athenaObj
```

```
ath = athenaObj.Interpreter()

ath.cmd("emo mp")

ath.cmd("tin a 45")
ath.cmd("tie t 0,20")
ath.cmd("tie a rb,.3,.3,.4,.8")
ath.cmd("tie r pt,(c,4),(bg,oc,(3,3,2)),(c,1)")

ath.cmd("tin b 65")
ath.cmd("tie t 10,20")
ath.cmd("tie a re,15,.3,1")
ath.cmd("tie r pt,(bg,rp,(2,1,1,1)),(c,1),(c,1)")

ath.cmd("tin c 67")
ath.cmd("tie t 15,25")
ath.cmd("tie a rb,.1,.1,.4,.6")
ath.cmd("tie r cs,(rb,.2,.2,.01,1.5)")

ath.cmd("eln")
ath.cmd("elh")
```

3.17. Musical Design Report 1

- Must be primarily rhythmic in nature
- Must employ at least 4 different timbre sources
- Should have at least an AB or ABA form
- Must prominently feature both the beta and exponential distributions
- Can be composed with athenaCL, athenaCL and other tools, or other tools alone
- See syllabus for details on other aspects

3.18. Digital Audio Workstations

- The merger of software for editing MIDI and notation with software for editing digital audio
- Numerous commercial varieties: ProTools, Digital Performer, Cubase, FL, Logic, GarageBand
- Inexpensive varieties: Reaper
- Free varieties: Ardour, Rosegarden
- Having access to a DAW with virtual instruments will greatly assist your projects in this class

3.19. Digital Audio Workstations: Importing and Mixing Digital Audio

- Create tracks to store audio

- Drag and drop digital audio into a track
- Adjust levels, process, and edit
- Bounce to disc to mix down to a single audio file

3.20. Digital Audio Workstations: Importing MIDI and Rendering Digital Audio

- Create tracks to store MIDI or for virtual instruments
- Drag and drop MIDI into a track
- Render, freeze, or bounce realization of virtual instrument