Axle

Lawful Scientific Computing for Scala

Adam Pingel
axle-lang.org

Contents

TEXT 145

Introduction

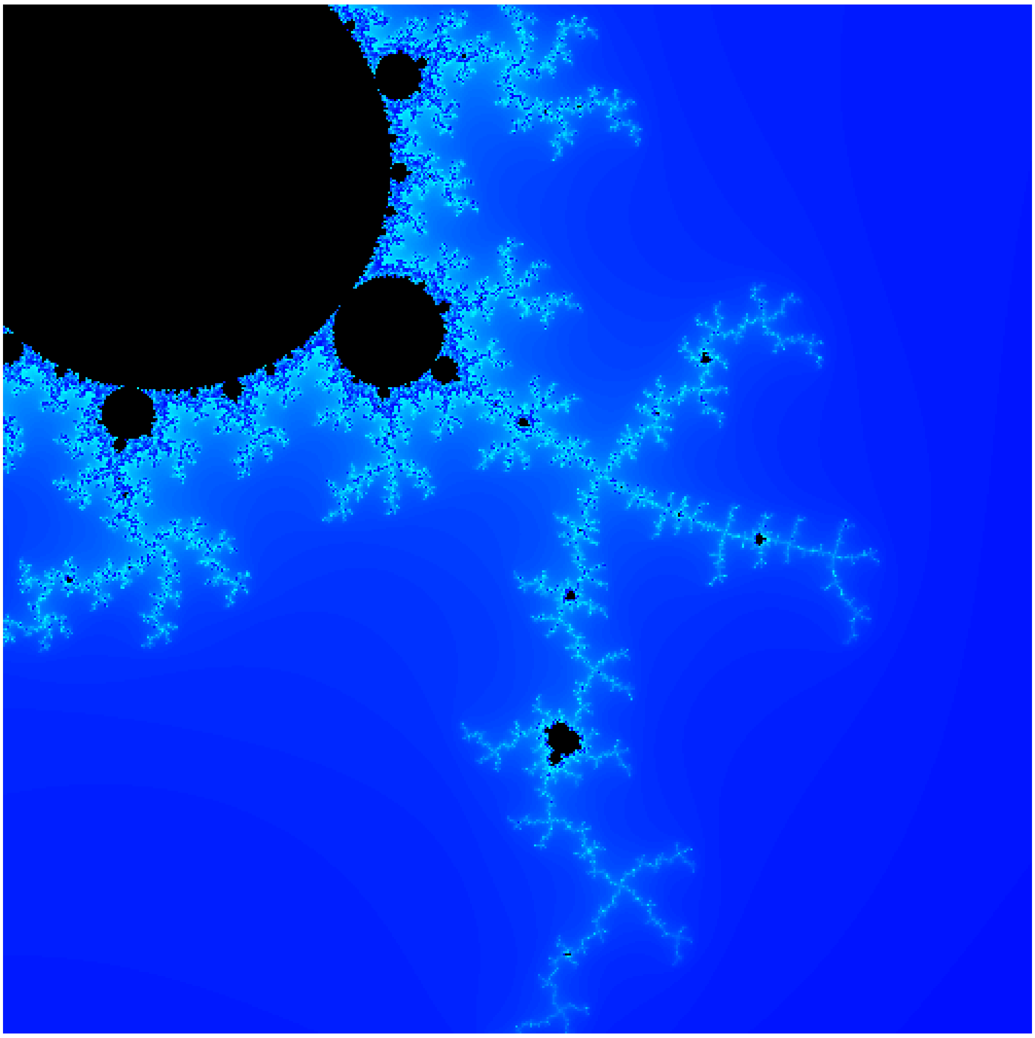
Objectives

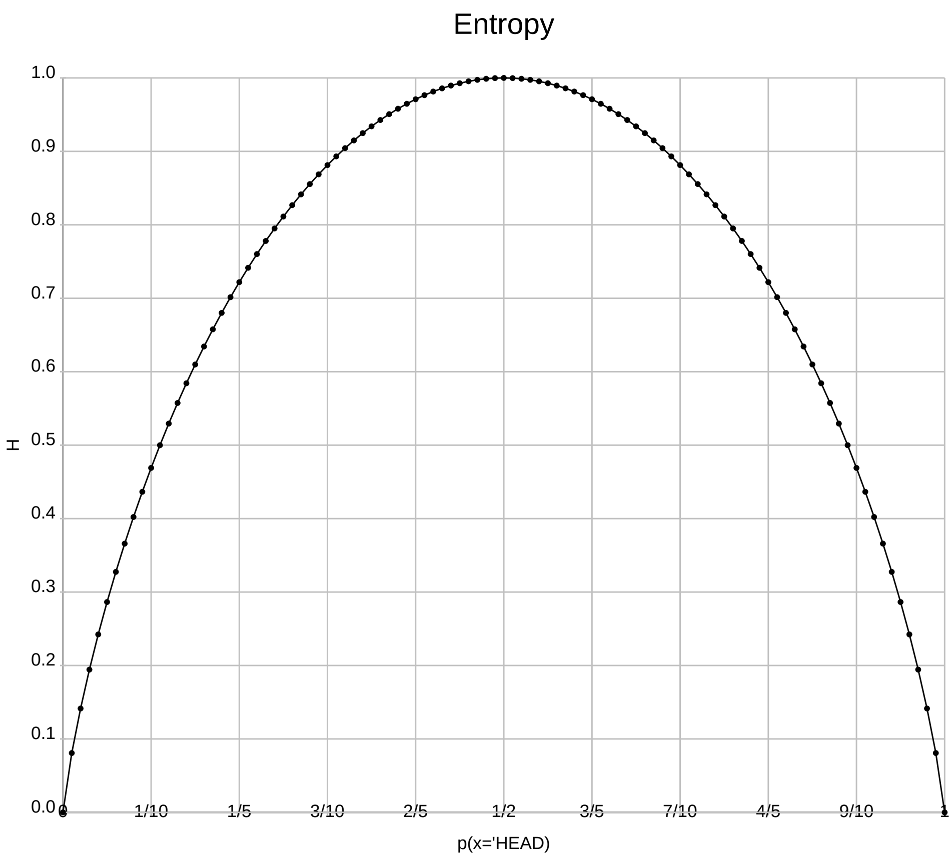
Practice coding in a strongly functional style and writing about it

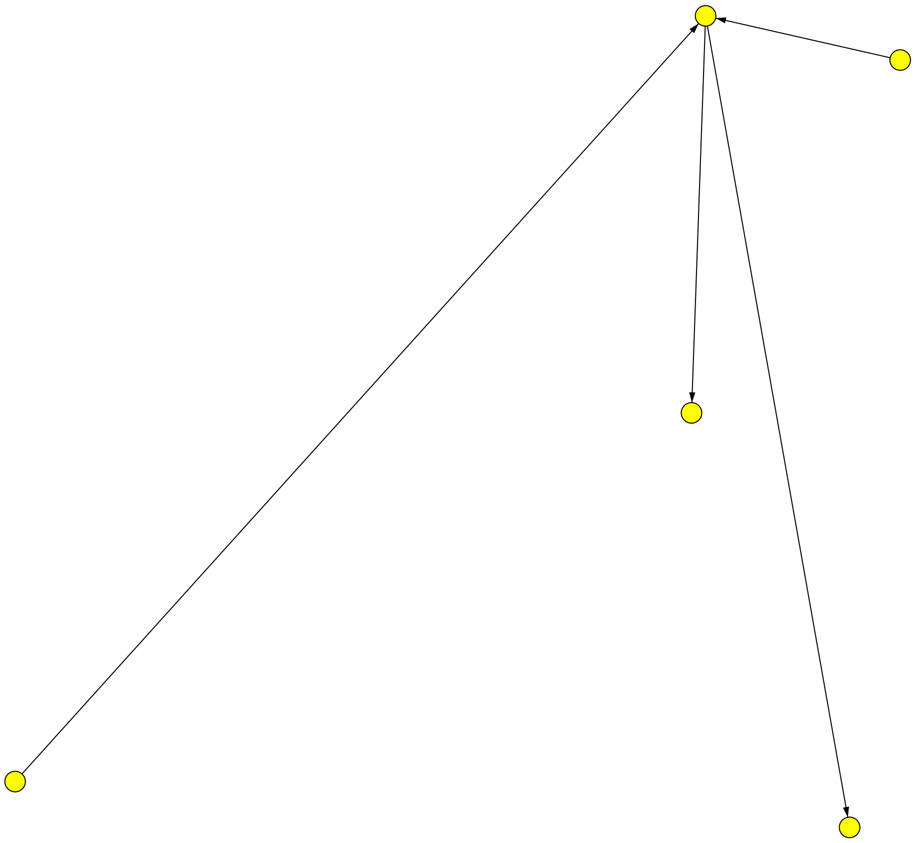
No doubles (easy path to "theorems for free")

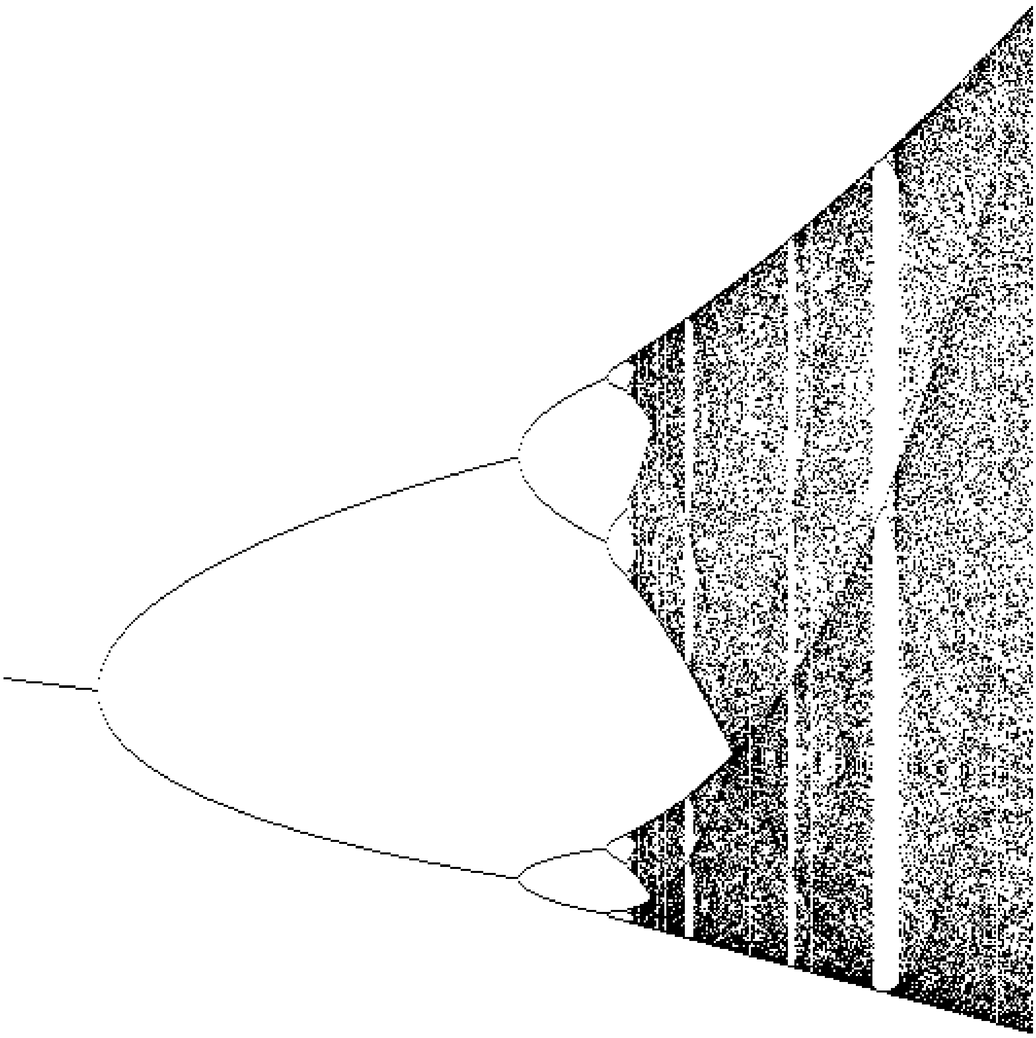
Lawful AI

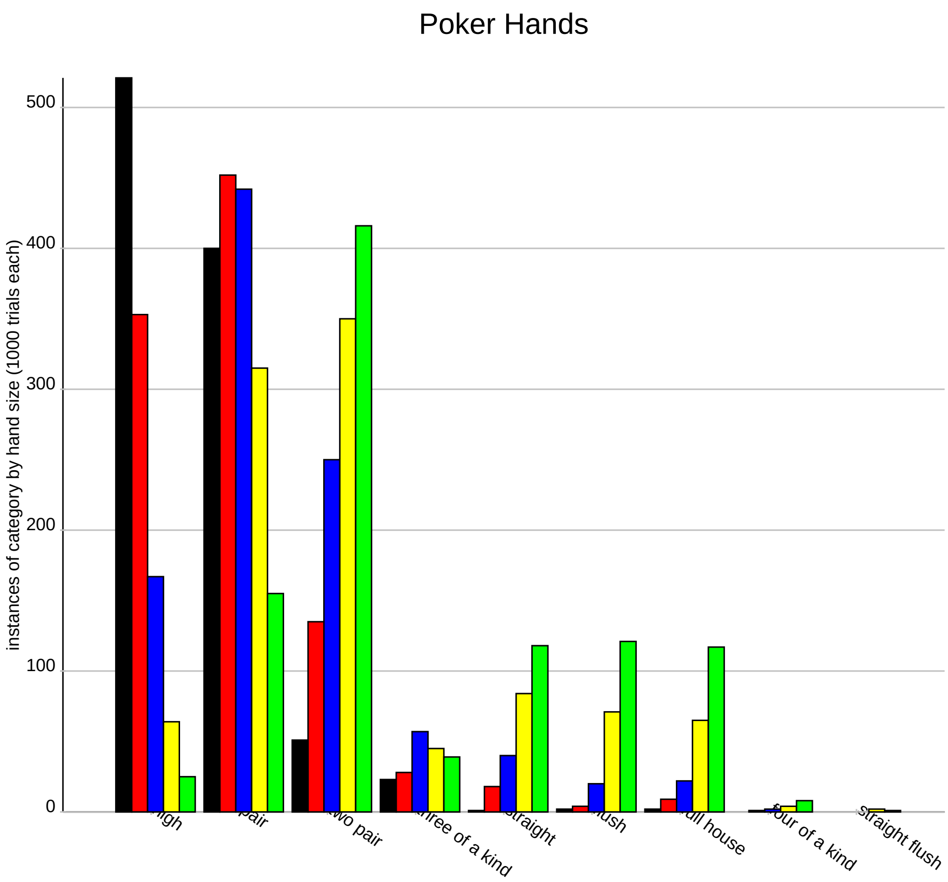
Gallery

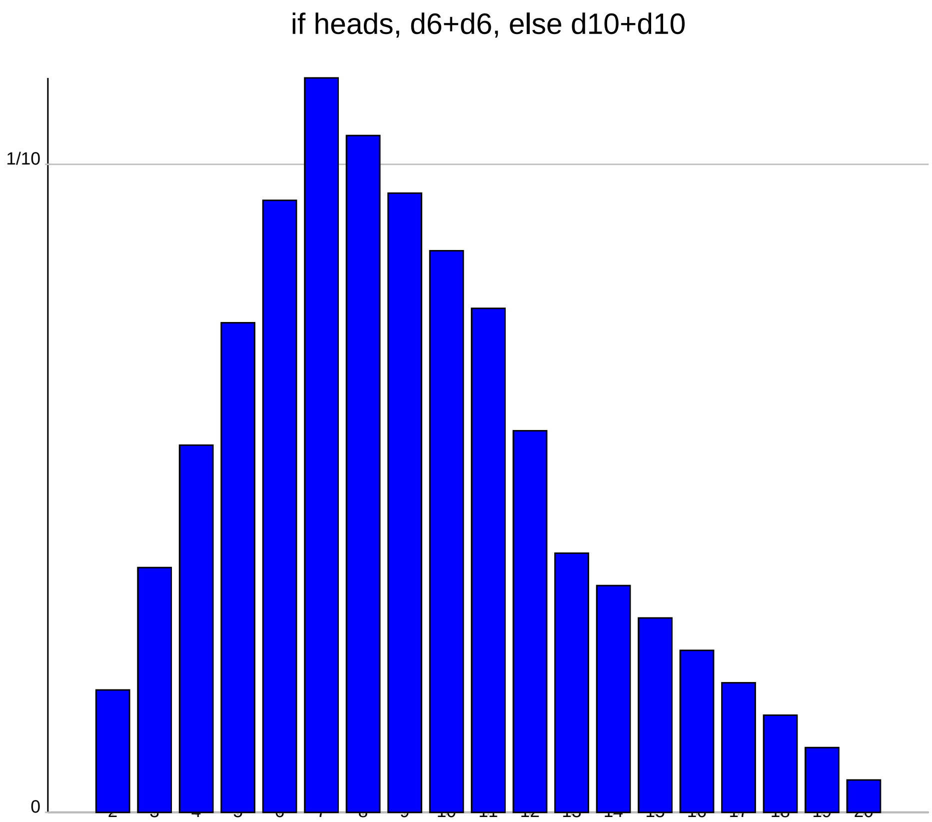


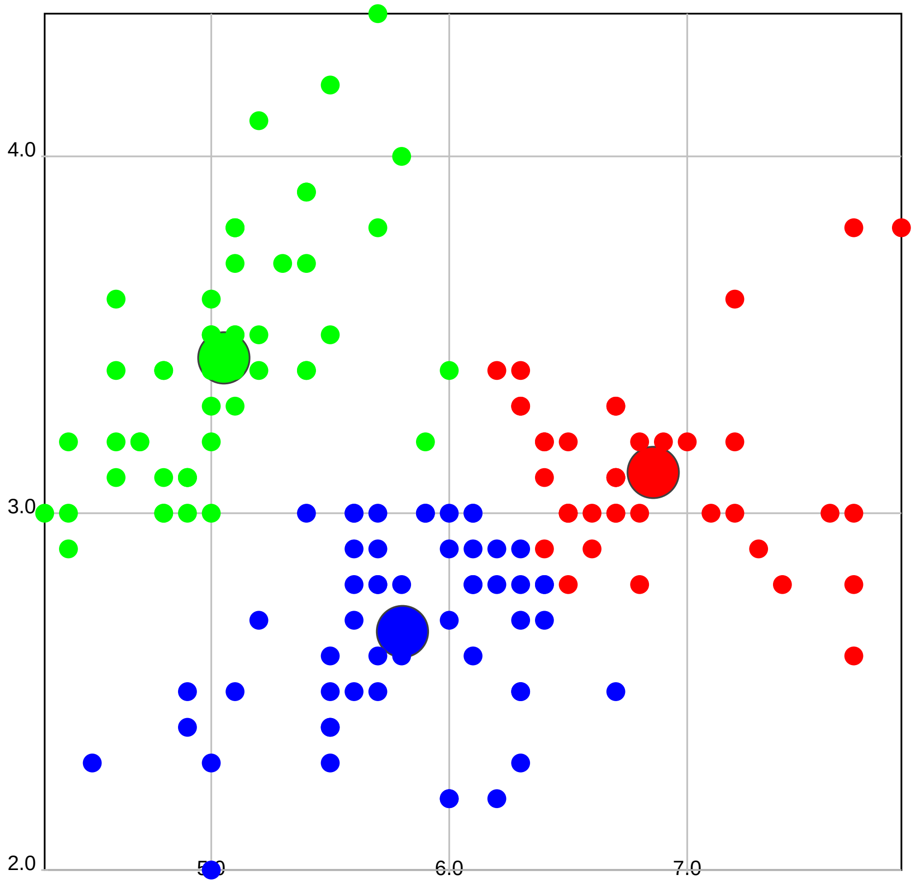

Entropy
H
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1/10
1/5
3/10
2/5
1/2
3/5
7/10
4/5
9/10
1
p(x='HEAD)

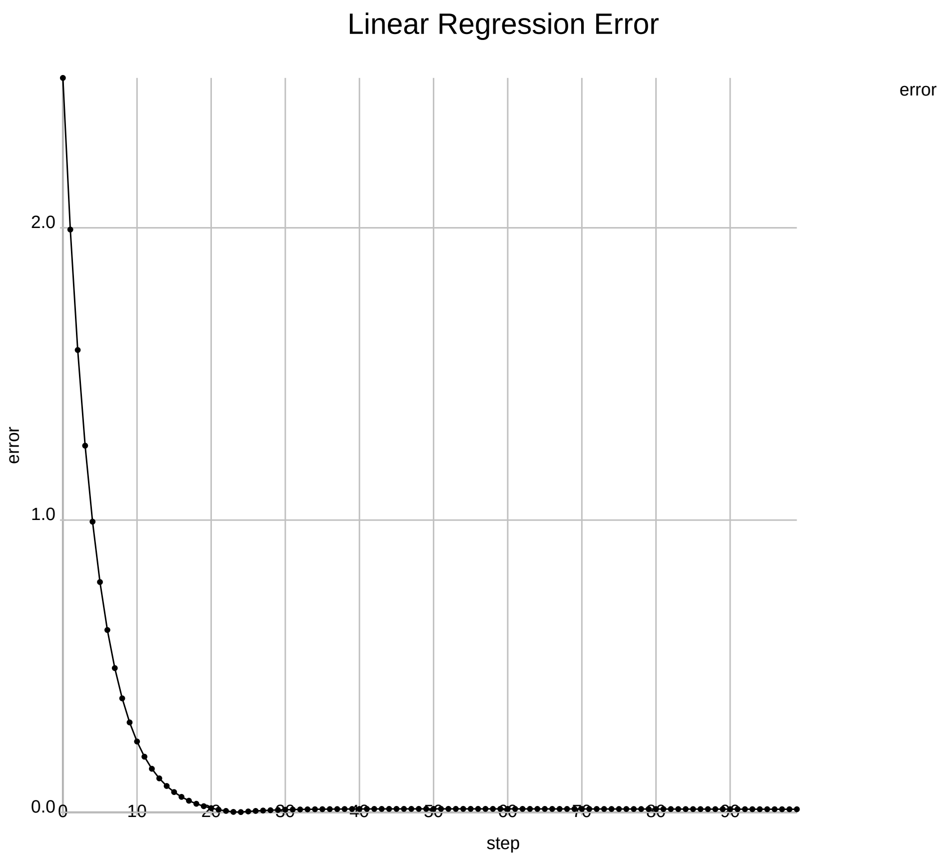




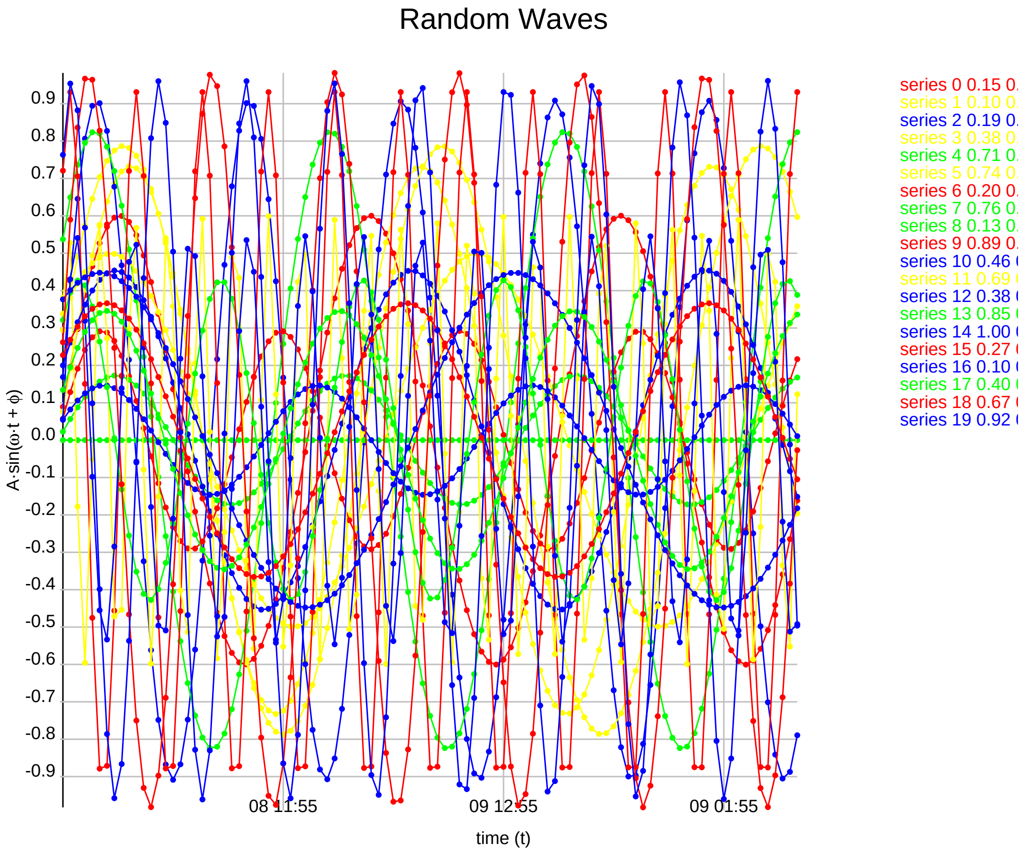

Poker Hands
instances of category by hand size (1000 trials each)
500
400
300
200
100
0
igh
pair
wo pair
hree of a kind
straight
lush
ull house
four of a kind
straight flush


if heads, d6+d6, else d10+d10
1/10
0
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20


4.0
3.0
2.0
5.0
6.0
7.0


Linear Regression Error
error
error
2.0
1.0
0.0
0
10
20
30
40
50
60
70
80
90
step


Greenland
Iceland
Finland
Sweden
Norway
United Kingdom
Poland
Germany
Ukraine
France
Italy
Spain
Turkey
Canada
United States
Mexico
North Atlantic Ocean
Algeria
Libya
Egypt
Iraq
Mali
Niger
Chad
Sudan
Nigeria
Ethiopia
Venezuela
Colombia



Installation

Axle as a dependency of an SBT project.

Install SBT

See **SBT**

Create SBT Project from Giter8 Template

```
sbt new axlelang/axle.g8
```

(Less commonly used `axle-laws`, `axle-awt`, and `axle-parallel` are not included in seed project.)

In addition to the axle-lang.org jars are several other third party jars. Axle is compiled with these jars in `provided` scope, meaning that they are compiled and packaged with the expectation that the user of the Axle jars will explicitly provide those dependencies.

Next Steps

Run `sbt console` to launch the Scala REPL with the Axle jars in the classpath. Axle works well interactively -- especially during prototyping, debugging, and testing. Any of the Axle tutorials can be copied and pasted into the REPL.

To start writing code, check out `src/main/scala/example/Hello.scala`, and go from there.

Releases

0.6.5 is the most recent released version:

See the **Road Map** for more information on the release schedule.

Snapshots

Snapshot versions are created for every commit and hosted on the **Sonatype snapshot repo**.

Community Resources

- Chat on the gitter channel: 

- **@axledsl** Twitter handle

Foundation

Data structures and functions

Functional

To be written

Scala

To be written...

Cats

To be written

Axle makes use of several Typelevel libraries including

- Cats
- Cats Effect
- Spire
- Monix
- ...

Architecture

Axle generally strives to follow the patterns established by the **Typelevel** projects.

With few exceptions, the functions are side-effect free.

The typeclass patterns are drawn from two traditions:

1. **Typeclassopedia**
2. Abstract Algebra

The algorithms are increasingly defined only in terms of these typeclasses. Concrete runtime implementations will require witnesses that map non-Axle data structures onto the typeclass methods and laws. Laws are organized into a separate `axle-laws` jar for use in tests by code that builds upon these typeclasses. Many such witnesses are provided by Axle for native Scala collections.

Witnesses are also defined for other common jars from the Java and Scala ecosystems. Read more about **"these third party libraries"**.

Remaining Design Issues

Please get in touch if you'd like to discuss these or other questions.

Package Objects

This page describes functions in `axle.logic` and `axle.math` package objects.

Imports

```
import cats.implicits._

import spire.algebra._

import axle.logic._
import axle.math._

implicit val rngInt: Rng[Int] = spire.implicits.IntAlgebra
implicit val ringLong: Ring[Long] = spire.implicits.LongAlgebra
implicit val boolBoolean: Bool[Boolean] = spire.implicits.BooleanStructure
```

Logic aggregators # and #:

```
#(List(1, 2, 3)) { i: Int => i % 2 == 0 }
// res1: Boolean = true

#(List(1, 2, 3)) { i: Int => i % 2 == 0 }
// res2: Boolean = false
```

Sum and multiply aggregators Σ and Π. Note that Σ and Π are also available in `spire.optional.unicode._`.

```
Σ((1 to 10) map { _ * 2 })
// res3: Int = 110

Π((1L to 10L) map { _ * 2 })
// res4: Long = 3715891200L
```

Doubles, triples, and cross-products

```
doubles(Set(1, 2, 3))
// res5: Seq[(Int, Int)] = List((1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2))

triples(Set(1, 2, 3))
// res6: Seq[(Int, Int, Int)] = List(
//   (1, 2, 3),
//   (1, 3, 2),
//   (2, 1, 3),
//   (2, 3, 1),
//   (3, 1, 2),
```

```
//   (3, 2, 1)
// )

#(List(1, 2, 3))(List(4, 5, 6)).toList
// res7: List[(Int, Int)] = List(
//   (1, 4),
//   (1, 5),
//   (1, 6),
//   (2, 4),
//   (2, 5),
//   (2, 6),
//   (3, 4),
//   (3, 5),
//   (3, 6)
// )
```

Powerset

```
#(0 until 4)
// res8: IndexedPowerSet[Int] = Iterable(
//   Set(),
//   Set(0),
//   Set(1),
//   Set(0, 1),
//   Set(2),
//   Set(0, 2),
//   Set(1, 2),
//   Set(0, 1, 2),
//   Set(3),
//   Set(0, 3),
//   Set(1, 3),
//   Set(0, 1, 3),
//   Set(2, 3),
//   Set(0, 2, 3),
//   Set(1, 2, 3),
//   Set(0, 1, 2, 3)
// )

val ps = #(Vector("a", "b", "c"))
// ps: IndexedPowerSet[String] = Iterable(
//   Set(),
//   Set("a"),
//   Set("b"),
//   Set("a", "b"),
//   Set("c"),
//   Set("a", "c"),
//   Set("b", "c"),
//   Set("a", "b", "c")
// )

ps.size
// res9: Int = 8
```

```
ps(7)
// res10: Set[String] = Set("a", "b", "c")
```

Permutations

```
permutations(0 until 4)(2).toList
// res11: List[IndexedSeq[Int]] = List(
//   Vector(0, 1),
//   Vector(0, 2),
//   Vector(0, 3),
//   Vector(1, 0),
//   Vector(1, 2),
//   Vector(1, 3),
//   Vector(2, 0),
//   Vector(2, 1),
//   Vector(2, 3),
//   Vector(3, 0),
//   Vector(3, 1),
//   Vector(3, 2)
// )
```

Combinations

```
combinations(0 until 4)(2).toList
// res12: List[IndexedSeq[Int]] = List(
//   Vector(0, 1),
//   Vector(0, 2),
//   Vector(0, 3),
//   Vector(1, 2),
//   Vector(1, 3),
//   Vector(2, 3)
// )
```

Indexed Cross Product

```
val icp = IndexedCrossProduct(Vector(
  Vector("a", "b", "c"),
  Vector("d", "e"),
  Vector("f", "g", "h")))
// icp: IndexedCrossProduct[String] = Iterable(
//   List("a", "d", "f"),
//   List("a", "d", "g"),
//   List("a", "d", "h"),
//   List("a", "e", "f"),
//   List("a", "e", "g"),
//   List("a", "e", "h"),
//   List("b", "d", "f"),
//   List("b", "d", "g"),
```

```
//   List("b", "d", "h"),
//   List("b", "e", "f"),
//   List("b", "e", "g"),
//   List("b", "e", "h"),
//   List("c", "d", "f"),
//   List("c", "d", "g"),
//   List("c", "d", "h"),
//   List("c", "e", "f"),
//   List("c", "e", "g"),
//   List("c", "e", "h")
// )

icp.size
// res13: Int = 18

icp(4)
// res14: Seq[String] = List("a", "e", "g")
```

Algebra

The spire project is a dependency of Axle. `spire.algebra` defines typeclasses for Monoid, Group, Ring, Field, VectorSpace, etc, and witnesses for many common numeric types as well as those defined in `spire.math`

The `axle.algebra` package defines several categories of typeclasses:

- higher-kinded: Functor, Finite, Indexed, Aggregatable
- mathematical: LinearAlgebra, LengthSpace
- visualization: Tics, Plottable

Axioms are defined in the axle.algebra.laws package as ScalaCheck properties.

They are organized with Discipline.

Linear Algebra

A `LinearAlgebra` typeclass.

The `axle-jblas` spoke provides witnesses for JBLAS matrices.

The default jblas matrix `toString` isn't very readable, so this tutorial wraps most results in the Axle `string` function, invoking the `cats.Show` witness for those matrices.

Imports and implicits

Import JBLAS and Axle's `LinearAlgebra` witness for it.

```
import cats.implicits._
```

```
import spire.algebra.Field
import spire.algebra.NRoot

import axle._
import axle.jblas._
import axle.syntax.linearalgebra.matrixOps

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
implicit val nrootDouble: NRoot[Double] = spire.implicits.DoubleAlgebra

implicit val laJblasDouble = axle.jblas.linearAlgebraDoubleMatrix[Double]
import laJblasDouble._
```

Creating Matrices

```
ones(2, 3).show
// res16: String = """1.000000 1.000000 1.000000
// 1.000000 1.000000 1.000000"""

ones(1, 4).show
// res17: String = "1.000000 1.000000 1.000000 1.000000"

ones(4, 1).show
// res18: String = """1.000000
// 1.000000
// 1.000000
// 1.000000"""
```

Creating matrices from arrays

```
fromColumnMajorArray(2, 2, List(1.1, 2.2, 3.3, 4.4).toArray).show
// res19: String = """1.100000 3.300000
// 2.200000 4.400000"""

fromColumnMajorArray(2, 2, List(1.1, 2.2, 3.3, 4.4).toArray).t.show
// res20: String = """1.100000 2.200000
// 3.300000 4.400000"""

val m = fromColumnMajorArray(4, 5, (1 to 20).map(_.toDouble).toArray)
// m: org.jblas.DoubleMatrix = [1.000000, 5.000000, 9.000000, 13.000000,
 17.000000; 2.000000, 6.000000, 10.000000, 14.000000, 18.000000; 3.000000,
 7.000000, 11.000000, 15.000000, 19.000000; 4.000000, 8.000000, 12.000000,
 16.000000, 20.000000]
m.show
// res21: String = """1.000000 5.000000 9.000000 13.000000 17.000000
// 2.000000 6.000000 10.000000 14.000000 18.000000
// 3.000000 7.000000 11.000000 15.000000 19.000000
// 4.000000 8.000000 12.000000 16.000000 20.000000"""
```

Random matrices

```
val r = rand(3, 3)
// r: org.jblas.DoubleMatrix = [0.784904, 0.991025, 0.576887; 0.511421,
 0.199427, 0.880304; 0.454225, 0.298775, 0.259226]

r.show
// res22: String = """0.784904 0.991025 0.576887
// 0.511421 0.199427 0.880304
// 0.454225 0.298775 0.259226"""
```

Matrices defined by functions

```
matrix(4, 5, (r, c) => r / (c + 1d)).show
// res23: String = """0.000000 0.000000 0.000000 0.000000 0.000000
// 1.000000 0.500000 0.333333 0.250000 0.200000
// 2.000000 1.000000 0.666667 0.500000 0.400000
// 3.000000 1.500000 1.000000 0.750000 0.600000"""

matrix(4, 5, 1d,
  (r: Int) => r + 0.5,
  (c: Int) => c + 0.6,
  (r: Int, c: Int, diag: Double, left: Double, right: Double) => diag).show
// res24: String = """1.000000 1.600000 2.600000 3.600000 4.600000
// 1.500000 1.000000 1.600000 2.600000 3.600000
// 2.500000 1.500000 1.000000 1.600000 2.600000
// 3.500000 2.500000 1.500000 1.000000 1.600000"""
```

Metadata

```
val x = fromColumnMajorArray(3, 1, Vector(4.0, 5.1, 6.2).toArray)
// x: org.jblas.DoubleMatrix = [4.000000; 5.100000; 6.200000]
x.show
// res25: String = """4.000000
// 5.100000
// 6.200000"""

val y = fromColumnMajorArray(3, 1, Vector(7.3, 8.4, 9.5).toArray)
// y: org.jblas.DoubleMatrix = [7.300000; 8.400000; 9.500000]
y.show
// res26: String = """7.300000
// 8.400000
// 9.500000"""

x.isEmpty
// res27: Boolean = false

x.isRowVector
// res28: Boolean = false
```

```
x.isColumnVector
// res29: Boolean = true

x.isSquare
// res30: Boolean = false

x.isScalar
// res31: Boolean = false

x.rows
// res32: Int = 3

x.columns
// res33: Int = 1

x.length
// res34: Int = 3
```

Accessing columns, rows, and elements

```
x.column(0).show
// res35: String = """4.000000
// 5.100000
// 6.200000"""

x.row(1).show
// res36: String = "5.100000"

x.get(2, 0)
// res37: Double = 6.2

val fiveByFive = fromColumnMajorArray(5, 5, (1 to 25).map(_.toDouble).toArray)
// fiveByFive: org.jblas.DoubleMatrix = [1.000000, 6.000000, 11.000000,
 16.000000, 21.000000; 2.000000, 7.000000, 12.000000, 17.000000, 22.000000;
 3.000000, 8.000000, 13.000000, 18.000000, 23.000000; 4.000000, 9.000000,
 14.000000, 19.000000, 24.000000; 5.000000, 10.000000, 15.000000, 20.000000,
 25.000000]

fiveByFive.show
// res38: String = """1.000000 6.000000 11.000000 16.000000 21.000000
// 2.000000 7.000000 12.000000 17.000000 22.000000
// 3.000000 8.000000 13.000000 18.000000 23.000000
// 4.000000 9.000000 14.000000 19.000000 24.000000
// 5.000000 10.000000 15.000000 20.000000 25.000000"""

fiveByFive.slice(1 to 3, 2 to 4).show
// res39: String = """12.000000 17.000000 22.000000
// 13.000000 18.000000 23.000000
// 14.000000 19.000000 24.000000"""

fiveByFive.slice(0.until(5,2), 0.until(5,2)).show
// res40: String = """1.000000 11.000000 21.000000
```

```
// 3.000000 13.000000 23.000000
// 5.000000 15.000000 25.000000"""
```

Negate, Transpose, Power

```
x.negate.show
// res41: String = """-4.000000
// -5.100000
// -6.200000"""

x.transpose.show
// res42: String = "4.000000 5.100000 6.200000"

// x.log
// x.log10

x.pow(2d).show
// res43: String = """16.000000
// 26.010000
// 38.440000"""
```

Mins, Maxs, Ranges, and Sorts

```
r.max
// res44: Double = 0.9910249644290331

r.min
// res45: Double = 0.19942686349533967

// r.ceil
// r.floor

r.rowMaxs.show
// res46: String = """0.991025
// 0.880304
// 0.454225"""

r.rowMins.show
// res47: String = """0.576887
// 0.199427
// 0.259226"""

r.columnMaxs.show
// res48: String = "0.784904 0.991025 0.880304"

r.columnMins.show
// res49: String = "0.454225 0.199427 0.259226"

rowRange(r).show
// res50: String = """0.414138
```

```
// 0.680877
// 0.194998"""

columnRange(r).show
// res51: String = "0.330680 0.791598 0.621078"

r.sortRows.show
// res52: String = """0.576887 0.784904 0.991025
// 0.199427 0.511421 0.880304
// 0.259226 0.298775 0.454225"""

r.sortColumns.show
// res53: String = """0.454225 0.199427 0.259226
// 0.511421 0.298775 0.576887
// 0.784904 0.991025 0.880304"""

r.sortRows.sortColumns.show
// res54: String = """0.199427 0.298775 0.454225
// 0.259226 0.511421 0.880304
// 0.576887 0.784904 0.991025"""
```

Statistics

```
r.rowMeans.show
// res55: String = """0.784272
// 0.530384
// 0.337409"""

r.columnMeans.show
// res56: String = "0.583517 0.496409 0.572139"

// median(r)

sumsq(r).show
// res57: String = "1.083946 1.111168 1.174932"

std(r).show
// res58: String = "0.144304 0.352090 0.253576"

cov(r).show
// res59: String = """0.013519 -0.011887 -0.000205
// -0.011887 0.029399 -0.020301
// -0.000205 -0.020301 0.032638"""

centerRows(r).show
// res60: String = """0.000632 0.206753 -0.207385
// -0.018963 -0.330957 0.349920
// 0.116816 -0.038634 -0.078182"""

centerColumns(r).show
// res61: String = """0.201388 0.494616 0.004748
// -0.072096 -0.296982 0.308165
```

```
// -0.129292 -0.197634 -0.312913"""

zscore(r).show
// res62: String = """1.395576 1.404799 0.018724
// -0.499608 -0.843483 1.215276
// -0.895968 -0.561316 -1.234000"""
```

Principal Component Analysis

```
val (u, s) = pca(r)
// u: org.jblas.DoubleMatrix = [-0.205393, -0.712497, 0.670941; 0.694886,
 0.376584, 0.612632; -0.689164, 0.592057, 0.417757]
// s: org.jblas.DoubleMatrix = [0.053047; 0.019972; 0.002538]

u.show
// res63: String = """-0.205393 -0.712497 0.670941
// 0.694886 0.376584 0.612632
// -0.689164 0.592057 0.417757"""

s.show
// res64: String = """0.053047
// 0.019972
// 0.002538"""
```

Horizontal and vertical concatenation

```
(x aside y).show
// res65: String = """4.000000 7.300000
// 5.100000 8.400000
// 6.200000 9.500000"""

(x atop y).show
// res66: String = """4.000000
// 5.100000
// 6.200000
// 7.300000
// 8.400000
// 9.500000"""
```

Addition and subtraction

```
val z = ones(2, 3)
// z: org.jblas.DoubleMatrix = [1.000000, 1.000000, 1.000000; 1.000000,
 1.000000, 1.000000]

z.show
// res67: String = """1.000000 1.000000 1.000000
// 1.000000 1.000000 1.000000"""
```

Matrix addition

```
import spire.implicits.additiveSemigroupOps

(z + z).show
// res68: String = """2.000000 2.000000 2.000000
// 2.000000 2.000000 2.000000"""
```

Scalar addition (JBLAS method)

```
z.addScalar(1.1).show
// res69: String = """2.100000 2.100000 2.100000
// 2.100000 2.100000 2.100000"""
```

Matrix subtraction

```
import spire.implicits.additiveGroupOps

(z - z).show
// res70: String = """0.000000 0.000000 0.000000
// 0.000000 0.000000 0.000000"""
```

Scalar subtraction (JBLAS method)

```
z.subtractScalar(0.2).show
// res71: String = """0.800000 0.800000 0.800000
// 0.800000 0.800000 0.800000"""
```

Multiplication and Division

Scalar multiplication

```
z.multiplyScalar(3d).show
// res72: String = """3.000000 3.000000 3.000000
// 3.000000 3.000000 3.000000"""
```

Matrix multiplication

```
import spire.implicits.multiplicativeSemigroupOps

(z * z.transpose).show
// res73: String = """3.000000 3.000000
// 3.000000 3.000000"""
```

Scalar division (JBLAS method)

```
z.divideScalar(100d).show
// res74: String = """0.010000 0.010000 0.010000
// 0.010000 0.010000 0.010000"""
```

Map element values

```
implicit val endo = axle.jblas.endoFunctorDoubleMatrix[Double]
// endo: algebra.Endofunctor[org.jblas.DoubleMatrix, Double] =
 axle.jblas.package$$anon$1@75b6c5b2
import axle.syntax.endofunctor.endofunctorOps

val half = ones(3, 3).map(_ / 2d)
// half: org.jblas.DoubleMatrix = [0.500000, 0.500000, 0.500000; 0.500000,
 0.500000, 0.500000; 0.500000, 0.500000, 0.500000]

half.show
// res75: String = """0.500000 0.500000 0.500000
// 0.500000 0.500000 0.500000
// 0.500000 0.500000 0.500000"""
```

Boolean operators

```
(r lt half).show
// res76: String = """0.000000 0.000000 0.000000
// 0.000000 1.000000 0.000000
// 1.000000 1.000000 1.000000"""

(r le half).show
// res77: String = """0.000000 0.000000 0.000000
// 0.000000 1.000000 0.000000
// 1.000000 1.000000 1.000000"""

(r gt half).show
// res78: String = """1.000000 1.000000 1.000000
// 1.000000 0.000000 1.000000
// 0.000000 0.000000 0.000000"""

(r ge half).show
// res79: String = """1.000000 1.000000 1.000000
// 1.000000 0.000000 1.000000
// 0.000000 0.000000 0.000000"""

(r eq half).show
// res80: String = """0.000000 0.000000 0.000000
// 0.000000 0.000000 0.000000
// 0.000000 0.000000 0.000000"""

(r ne half).show
// res81: String = """1.000000 1.000000 1.000000
// 1.000000 1.000000 1.000000
```

```
// 1.000000 1.000000 1.000000"""

((r lt half) or (r gt half)).show
// res82: String = """1.000000 1.000000 1.000000
// 1.000000 1.000000 1.000000
// 1.000000 1.000000 1.000000"""

((r lt half) and (r gt half)).show
// res83: String = """0.000000 0.000000 0.000000
// 0.000000 0.000000 0.000000
// 0.000000 0.000000 0.000000"""

((r lt half) xor (r gt half)).show
// res84: String = """1.000000 1.000000 1.000000
// 1.000000 1.000000 1.000000
// 1.000000 1.000000 1.000000"""

((r lt half) not).show
// res85: String = """1.000000 1.000000 1.000000
// 1.000000 0.000000 1.000000
// 0.000000 0.000000 0.000000"""
```

Higher order methods

```
(m.map(_ + 1)).show
// res86: String = """2.000000 6.000000 10.000000 14.000000 18.000000
// 3.000000 7.000000 11.000000 15.000000 19.000000
// 4.000000 8.000000 12.000000 16.000000 20.000000
// 5.000000 9.000000 13.000000 17.000000 21.000000"""

(m.map(_ * 10)).show
// res87: String = """10.000000 50.000000 90.000000 130.000000 170.000000
// 20.000000 60.000000 100.000000 140.000000 180.000000
// 30.000000 70.000000 110.000000 150.000000 190.000000
// 40.000000 80.000000 120.000000 160.000000 200.000000"""

// m.foldLeft(zeros(4, 1))(_ + _)

(m.foldLeft(ones(4, 1))(_ mulPointwise _)).show
// res88: String = """9945.000000
// 30240.000000
// 65835.000000
// 122880.000000"""

// m.foldTop(zeros(1, 5))(_ + _)

(m.foldTop(ones(1, 5))(_ mulPointwise _)).show
// res89: String = "24.000000 1680.000000 11880.000000 43680.000000
 116280.000000"
```

Logic

Conjunctive Normal Form Converter

Imports

```
import cats.implicits._
import axle.logic.FirstOrderPredicateLogic._
```

Example CNF conversion

```
import axle.logic.example.SamplePredicates._

val z = Symbol("z")

val s = #(z # Z, (A(z) # G(z)) # (B(z) # H(z)))

val (cnf, skolemMap) = conjunctiveNormalForm(s)
```

```
cnf.show
// res91: String = "((¬A(Symbol(sk0)) # (¬G(Symbol(sk1)) # (B(Symbol(sk2)) #
 H(Symbol(sk3))))) # (((¬B(Symbol(sk4)) # ¬H(Symbol(sk5))) # A(Symbol(sk6))) #
 ((¬B(Symbol(sk4)) # ¬H(Symbol(sk5))) # G(Symbol(sk7)))))"

skolemMap
// res92: Map[Symbol, Set[Symbol]] = HashMap(
//   'sk2 -> Set(),
//   'sk6 -> Set(),
//   'sk7 -> Set(),
//   'sk3 -> Set(),
//   'sk4 -> Set(),
//   'sk1 -> Set(),
//   'sk0 -> Set(),
//   'sk5 -> Set()
// )
```

Support for Third Party Libraries

Witnesses for 3rd party libraries.

Parallel Collections

```
"org.axle-lang" %% "axle-parallel" % "0.6.5"
```

For use with Scala **Parallel Collections** library (`"org.scala-lang.modules" %% "scala-parallel-collections" % ...`)

XML

```
"org.axle-lang" %% "axle-xml" % "0.6.5"
```

For use with Scala **XML** library (`"org.scala-lang.modules" %% "scala-xml" % ...`)

XML includes `axle.web`, where HTML and SVG visualizations reside.

JBLAS

```
"org.axle-lang" %% "axle-jblas" % "0.6.5"
```

Linear Algebra and other witnesses for **JBLAS** which itself is a wrapper for **LAPACK**. Includes Principal Component Analysis (PCA).

JODA

```
"org.axle-lang" %% "axle-joda" % "0.6.5"
```

Witnesses for the **Joda** time library.

JUNG

```
"org.axle-lang" %% "axle-jung" % "0.6.5"
```

Directed and Undirected **Graph** witnesses for the **JUNG** library.

AWT

```
"org.axle-lang" %% "axle-awt" % "0.6.5"
```

Witnesses for **AWT**

Future Work

Scala 3

- Scala 3
- convert to scalameta munit
- correct "Package Objects" doc

Bugs and adoption barriers

- Fix `LogisticRegression` and move `LogisticRegression.md` back
- Fix `GeneticAlgorithmSpec`
- Featurizing functions should return HLists or other typelevel sequences in order to avoid being told the number of features
- Redo Logic using Abstract Algebra
- Simple graph implementation so that `axle-core` can avoid including `axle-jung`
- `svgJungDirectedGraphVisualization` move to a `axle-jung-xml` jar?
- Will require externalizing the layout to its own.... typeclass?
- Layout of bayesian network is quite bad -- check ABE SVG
- `axle-png` to avoid Xvfb requirement during tests
- Chicklet borders / colors on site
- Factor `axle.algebra.chain` in terms of well-known combinators

Types and Axioms

- Replace `Finite` with Shapeless's version (eg `Sized[Vector[_], nat.2]`)
- Delete `Finite` conversions for jung
- Replace with Cats: `FoldLeft`, `Bijection`, `FunctionPair`, `Endofunctor`
- Define laws for `Scanner`, `Aggregator`, `Zipper`, `Indexed`, `Talliable`, `Finite`?
- Sort out `MapFrom`, `FromStream`, `FromSet`
- Test `axle.algebra.tuple2Field`
- `similarity` syntax for `SimilaritySpace` (see `axle.bio.*`)
- Projections of jung graphs for `Finite`
- kittens or magnolia
- pattern match in `FirstOrderPredicateLogic`
- subtyping for `Suit` and `Rank`
- Machinist?

- Type-level matrix dimension using `-Yliteral-types` and `singleton-ops` in `LinearAlgebra` typeclass
- Make the `Int` abstract in `KMeans{,Visualization}`, `LinearAlgebra`, etc
- Eigenvectors
- # # # # # means "sums are left adjoint to diagonals, which are left adjoint to products."

Compute Engines

- Bring back Spark spoke -- Solve the Spark ClassTag issue (see Frameless?)
- Performance benchmarking
- netlib-java Matrix
- GPU/CUDA support
- Algebird/Scalding for distributed matrices, HyperLogLog, etc
- Most MapRedicible witnesses are inefficient (eg calling toVector, toSeq, etc)

Hygiene

- Get rid of implicit arg passing to KMeans in `ClusterIrises.md` (and KMeansSpecification)
- Factor tics and tics-{joda,algebra,spire} into separate libs?
- remove unnecessary implicit Field, R{,i}ng, {Additive, Multiplicative}Monoid once spire/cats play well
- Fix "unreachable" default pattern match cases
- Review remaining usage of: `asInstanceOf`, `ClassTag`, and `Manifest`
- Review `groupBy` uses -- they use university equality. Replace with `Eq`
- axle.algorithms coverage > 80%
- axle.core coverage > 80%
- Rm throws from axle.jung
- Rm throws from axle.pgm.BayesianNetwork

Site

Near term / minor

- General
    - Expand acronyms and include wikipedia links in "Future Work" sections

    - Make dependencies clear in each section

- Introduction
    - Smaller images for Gallery
- Foundation
    - Architecture
- Math
    - Intro section bullets not nesting
- Random, Uncertain
    - Bayesian network rendering is missing tables
- Text
    - Say more about Python Grammar

Later

- `laikaIncludeAPI := true` in `build.sbt`
- look at more of **these options**
- Meta tag with keywords: axle, scala, dsl, data, analysis, science, open-source, adam pingel
- update google analytics version
- test animation with monix 3.4.0
- Friend of Spire
- README: data sets from `axle.data` (Astronomy, Evolution, Federalist Papers, Irises)

Build

- ghpagesCleanSite leaving stale files?
- GitHub "Releases" in sidebar should show "latest"
- keep axle.g8 and axle versions in sync
- site publish (`git push`) via github action?
- make axle.g8 more axle-flavored (use cats.IO App as parent for HelloWorld)

Math

Pythagorean Means

Arithmetic, Geometric, and Harmonic Means are all 'Pythagorean'.

See the wikipedia page on Pythagorean Means for more.

Arithmetic, Geometric, and Harmonic Mean Examples

Imports

```
import cats.implicits._

import spire.math.Real
import spire.algebra.Field
import spire.algebra.NRoot

import axle.math._

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
implicit val nrootDouble: NRoot[Double] = spire.implicits.DoubleAlgebra
```

Examples

Arithmetic mean

```
arithmeticMean(List(2d, 3d, 4d, 5d))
// res0: Double = 3.5
```

Geometric mean

```
geometricMean[Real, List](List(1d, 5d, 25d))
// res1: Real = Inexact(
//   f = spire.math.Real$$Lambda$10520/0x00000008026d0000@4843e38d
// )
```

Harmonic mean

```
harmonicMean(List(2d, 3d, 4d, 5d))
// res2: Double = 3.116883116883117
```

Generalized Mean

See the wikipedia page on Generalized Mean.

When the parameter **p** is 1, it is the arithmetic mean.

```
generalizedMean[Double, List](1d, List(2d, 3d, 4d, 5d))
// res3: Double = 3.5
```

As **p** approaches 0, it is the geometric mean.

```
generalizedMean[Double, List](0.0001, List(1d, 5d, 25d))
// res4: Double = 5.00043173370165
```

At **-1** it is the harmonic mean.

```
generalizedMean[Double, List](-1d, List(2d, 3d, 4d, 5d))
// res5: Double = 3.116883116883117
```

Moving means

```
import spire.math._
```

Moving arithmetic mean

```
movingArithmeticMean[List, Int, Double](
    (1 to 100).toList.map(_.toDouble),
    5)
// res6: List[Double] = List(
//   3.0,
//   4.0,
//   5.0,
//   6.0,
//   7.0,
//   8.0,
//   9.0,
//   10.0,
//   11.0,
//   12.0,
//   13.0,
//   14.0,
//   15.0,
// ...
```

Moving geometric mean

```
movingGeometricMean[List, Int, Real](
    List(1d, 5d, 25d, 125d, 625d),
    3)
// res7: List[Real] = List(
//   Inexact(f = spire.math.Real$$Lambda$10520/0x00000008026d0000@41277929),
//   Inexact(f = spire.math.Real$$Lambda$10525/0x00000008027745b0@2172af86),
//   Inexact(f = spire.math.Real$$Lambda$10525/0x00000008027745b0@5cc0d8cd)
// )
```

Moving harmonic mean

```
movingHarmonicMean[List, Int, Real](
    (1 to 5).toList.map(v => Real(v)),
    3)
// res8: List[Real] = List(
//   Exact(n = 18/11),
//   Exact(n = 36/13),
//   Exact(n = 180/47)
// )
```

Mean Average Precision at K

See the page on **mean average precision** at Kaggle

```
import spire.math.Rational
import axle.ml.RankedClassifierPerformance._
```

Examples (from **benhamner/Metrics**)

```
meanAveragePrecisionAtK[Int, Rational](List(1 until 5), List(1 until 5), 3)
// res10: Rational = 1
```

```
meanAveragePrecisionAtK[Int, Rational]
(List(List(1, 3, 4), List(1, 2, 4), List(1, 3)), List(1 until 6, 1 until 6, 1 until 6), 3)
// res11: Rational = 37/54
```

```
meanAveragePrecisionAtK[Int, Rational]
(List(1 until 6, 1 until 6), List(List(6, 4, 7, 1, 2), List(1, 1, 1, 1, 1)), 5)
// res12: Rational = 13/50
```

```
meanAveragePrecisionAtK[Int, Rational]
(List(List(1, 3), List(1, 2, 3), List(1, 2, 3)), List(1 until 6, List(1, 1, 1), List(1, 2, 1)),
// res13: Rational = 11/18
```

Pi

Two estimators for π

```
import axle.math._
```

Wallis

The first is attributed to Englishman John Wallis (1616 - 1703) who published this function in 1655. It is quite slow.

```
wallisΠ(100).toDouble
// res15: Double = 3.1337874906281624

wallisΠ(200).toDouble
// res16: Double = 3.137677900950936

wallisΠ(400).toDouble
// res17: Double = 3.1396322219293964

wallisΠ(800).toDouble
// res18: Double = 3.1406116723489452

wallisΠ(1600).toDouble
// res19: Double = 3.1411019714193746

wallisΠ(3200).toDouble
// res20: Double = 3.141347264592393
```

Monte Carlo

```
import cats.implicits._
import spire.algebra.Field

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
```

See the Wikipedia page on **Monte Carlo Methods**

This particular implementation requires that the number of trials be passed as a type `F` such that witnesses for typeclasses `Aggregatable`, `Finite`, and `Functor` are available in implicit scope.

While this may may seem initially over-engineered, it allows `F` as varied as `List` and Spark's `RDD` to be used to represent the number of trials and support the Monte Carlo simulation and resulting aggregation.

```
monteCarloPiEstimate((1 to 10000).toList, (n: Int) => n.toDouble)
// res21: Double = 3.1428
```

Fibonacci

```
import axle.math._
```

Linear using `foldLeft`

```
fibonacciByFold(10)
// res23: Int = 89
```

Recursive

```
fibonacciRecursively(10)
// res24: Int = 89
```

Some alternatives that are not in Axle include

Recursive with memoization

```
val memo = collection.mutable.Map(0 -> 0L, 1 -> 1L)

def fibonacciRecursivelyWithMemo(n: Int): Long = {
  if (memo.contains(n)) {
    memo(n)
  } else {
    val result = fibonacciRecursivelyWithMemo(n - 2)
 + fibonacciRecursivelyWithMemo(n - 1)
    memo += n -> result
    result
  }
}
```

```
fibonacciRecursivelyWithMemo(10)
// res25: Long = 55L
```

Recursive squaring

Imports

```
import org.jblas.DoubleMatrix

import cats.implicits._

import spire.algebra.EuclideanRing
import spire.algebra.NRoot
import spire.algebra.Rng
```

```
import axle._
import axle.jblas._

implicit val eucRingInt: EuclideanRing[Int] = spire.implicits.IntAlgebra
implicit val rngDouble: Rng[Double] = spire.implicits.DoubleAlgebra
implicit val nrootDouble: NRoot[Double] = spire.implicits.DoubleAlgebra
implicit val laJblasDouble = axle.jblas.linearAlgebraDoubleMatrix[Double]
import laJblasDouble._
```

The fibonacci sequence at N can be generated by taking the Nth power of a special 2x2 matrix. By employing the general-purpose strategy for exponentiation called "recursive squaring", we can achieve sub-linear time.

```
val base = fromColumnMajorArray(2, 2, List(1d, 1d, 1d, 0d).toArray)

def fibonacciSubLinear(n: Int): Long = n match {
  case 0 => 0L
  case _ => exponentiateByRecursiveSquaring(base, n).get(0, 1).toLong
}
```

Demo:

```
fibonacciSubLinear(78)
// res26: Long = 8944394323791464L
```

Note: Beyond 78 inaccuracies creep in due to the limitations of the `Double` number type.

Ackermann

See the Wikipedia page on the Ackermann function

```
import axle.math._
```

The computational complexity is enormous. Only for very small `m` and `n` can the function complete:

```
ackermann(1, 1)
// res28: Long = 3L

ackermann(3, 3)
// res29: Long = 61L
```

Future Work

- Collatz Conjecture vis
- Demo Mandelbrot with Rational

- Scrutinize `axle.math` and move out less reusable functions
- Complex Analysis
- Topoi
- N Queens
- Connection between dynamic programming and semiring
- Fourier transformations
- Blockchain
- Rainbow Tables

Units of Measurement

Quanta

Quanta, Units, and Conversions

`UnittedQuantity` is the primary case class in `axle.quanta`

The `axle.quanta` package models units of measurement. Via typeclasses, it implements expected operators like `+`, `-`, a unit conversion operator `in`, and a right associative value constructor `*:`

The "quanta" are Acceleration, Area, Angle, Distance, Energy, Flow, Force, Frequency, Information, Mass, Money, MoneyFlow, MoneyPerForce, Power, Speed, Temperature, Time, and Volume. Axle's values are represented in such a way that a value's "quantum" is present in the type, meaning that nonsensical expressions like `mile + gram` can be rejected at compile time.

Additionally, various values within the Quantum objects are imported. This package uses the definition of "Quantum" as "something that can be quantified or measured".

```
import axle._
import axle.quanta._
import axle.jung._
```

Quanta each define a Wikipedia link where you can find out more about relative scale:

```
Distance().wikipediaUrl
// res0: String = "http://en.wikipedia.org/wiki/Orders_of_magnitude_(length)"
```

A visualization of the Units of Measurement for a given Quantum can be produced by first creating the converter:

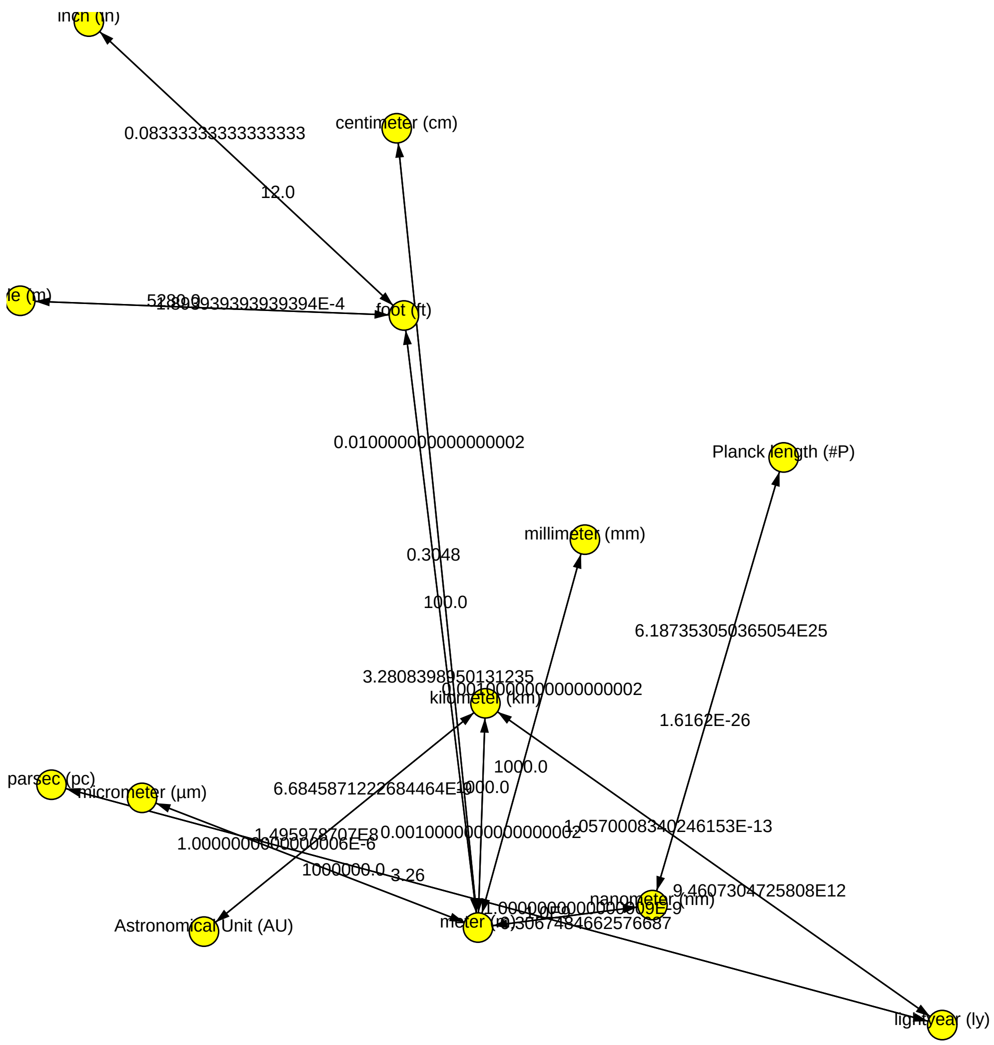
```
import edu.uci.ics.jung.graph.DirectedSparseGraph
import cats.implicits._
import spire.algebra.Field
import axle.algebra.modules.doubleRationalModule

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
implicit val distanceConverter
 = Distance.converterGraphK2[Double, DirectedSparseGraph]
```

Create a `DirectedGraph` visualization for it.

```
import cats.Show

implicit val showDDAt1 = new Show[Double => Double] {
  def show(f: Double => Double): String = f(1d).toString
}

import axle.visualize._

val dgVis =
  DirectedGraphVisualization[
    DirectedSparseGraph[UnitOfMeasurement[Distance],Double => Double],
    UnitOfMeasurement[Distance], Double => Double]
(distanceConverter.conversionGraph)
```

Render to an SVG.

```
import axle.web._
import cats.effect._

dgVis.svg[IO]("docwork/images/Distance.svg").unsafeRunSync()
```



Units

A conversion graph must be created with type parameters specifying the numeric type to be used in unitted quantity, as well as a directed graph type that will store the conversion graph. The conversion graphs should be placed in implicit scope. Within each are defined units of measurement which can be imported.

```
implicit val massConverter = Mass.converterGraphK2[Double, DirectedSparseGraph]
import massConverter._

implicit val powerConverter
 = Power.converterGraphK2[Double, DirectedSparseGraph]
import powerConverter._

import axle.algebra.modules.doubleRationalModule

// reuse distanceConverter defined in preceding section
import distanceConverter._

implicit val timeConverter = Time.converterGraphK2[Double, DirectedSparseGraph]
```

```
import timeConverter._
```

Standard Units of Measurement are defined:

```
gram
// res2: UnitOfMeasurement[Mass] = UnitOfMeasurement(
//   name = "gram",
//   symbol = "g",
//   wikipediaUrl = None
// )

foot
// res3: UnitOfMeasurement[Distance] = UnitOfMeasurement(
//   name = "foot",
//   symbol = "ft",
//   wikipediaUrl = None
// )

meter
// res4: UnitOfMeasurement[Distance] = UnitOfMeasurement(
//   name = "meter",
//   symbol = "m",
//   wikipediaUrl = None
// )
```

Construction

Values with units are constructed with the right-associative `*:` method on any spire `Number` type as long as a spire `Field` is implicitly available.

```
10d *: gram

3d *: lightyear

5d *: horsepower

3.14 *: second

200d *: watt
```

Show

A witness for the `cats.Show` typeclass is defined. `show` will return a `String` representation.

```
import cats.implicits._

(10d *: gram).show
// res10: String = "10.0 g"
```

Conversion

A Quantum defines a directed graph, where the UnitsOfMeasurement are the vertices, and the Conversions define the directed edges. See **Graph Theory** for more on how graphs work.

Quantities can be converted into other units of measurement. This is possible as long as 1) the values are in the same Quantum, and 2) there is a path in the Quantum between the two.

```
(10d *: gram in kilogram).show
// res11: String = "0.010000000000000002 Kg"
```

Converting between quanta is not allowed, and is caught at compile time:

```
(1 *: gram) in mile
// error: type mismatch;
//  found   : axle.quanta.UnitOfMeasurement[axle.quanta.Distance]
//  required: axle.quanta.UnitOfMeasurement[axle.quanta.Mass]
// (1 *: gram) in mile
//                ^^^^
```

Math

Addition and subtraction are defined on Quantity by converting the right Quantity to the unit of the left.

```
import spire.implicits.additiveGroupOps

((7d *: mile) - (123d *: foot)).show
// res13: String = "36837.0 ft"
```

```
{
  import spire.implicits._
  ((1d *: kilogram) + (10d *: gram)).show
}
// res14: String = "1010.0 g"
```

Addition and subtraction between different quanta is rejected at compile time:

```
(1d *: gram) + (2d *: foot)
// error: type mismatch;
//  found   : String
//  required: Int
//   ((1d *: kilogram) + (10d *: gram)).show
//   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
// error: type mismatch;
//  found   : axle.quanta.UnittedQuantity[axle.quanta.Distance,Double]
//  required: String
// (1d *: gram) + (2d *: foot)
//                  ^^^^^^^
```

Scalar multiplication comes from Spire's `CModule` typeclass:

```
import spire.implicits.rightModuleOps

((5.4 *: second) :* 100d).show
// res16: String = "540.0 s"
```

```
((32d *: century) :* (1d/3)).show
// res17: String = "10.666666666666666 century"
```

Unitted Trigonometry

Versions of the trigonometric functions sine, cosine, and tangent, require that the arguments are Angles.

Preamble

Imports, implicits, etc

```
import edu.uci.ics.jung.graph.DirectedSparseGraph

import cats.implicits._

import spire.algebra.Field
import spire.algebra.Trig

import axle.math._
import axle.quanta.Angle
import axle.quanta.UnitOfMeasurement
import axle.algebra.modules.doubleRationalModule
import axle.jung.directedGraphJung

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
implicit val trigDouble: Trig[Double] = spire.implicits.DoubleAlgebra

implicit val angleConverter
 = Angle.converterGraphK2[Double, DirectedSparseGraph]
```

```
import angleConverter.degree
import angleConverter.radian
```

Examples

```
cosine(10d *: degree)
// res19: Double = 0.984807753012208

sine(3d *: radian)
// res20: Double = 0.1411200080598672

tangent(40d *: degree)
// res21: Double = 0.8390996311772799
```

Geo Coordinates

Imports and implicits

```
import edu.uci.ics.jung.graph.DirectedSparseGraph

import cats.implicits._

import spire.algebra.Field
import spire.algebra.Trig
import spire.algebra.NRoot

import axle._
import axle.quanta._
import axle.algebra.GeoCoordinates
import axle.jung.directedGraphJung
import axle.algebra.modules.doubleRationalModule

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
implicit val trigDouble: Trig[Double] = spire.implicits.DoubleAlgebra
implicit val nrootDouble: NRoot[Double] = spire.implicits.DoubleAlgebra

implicit val angleConverter
 = Angle.converterGraphK2[Double, DirectedSparseGraph]
import angleConverter.°
```

Locations of SFO and HEL airports:

```
val sfo = GeoCoordinates(37.6189 *: °, 122.3750 *: °)
```

```
sfo.show
// res23: String = "37.6189° N 122.375° W"
```

```
val hel = GeoCoordinates(60.3172 *: °, -24.9633 *: °)
```

```
hel.show
// res24: String = "60.3172° N -24.9633° W"
```

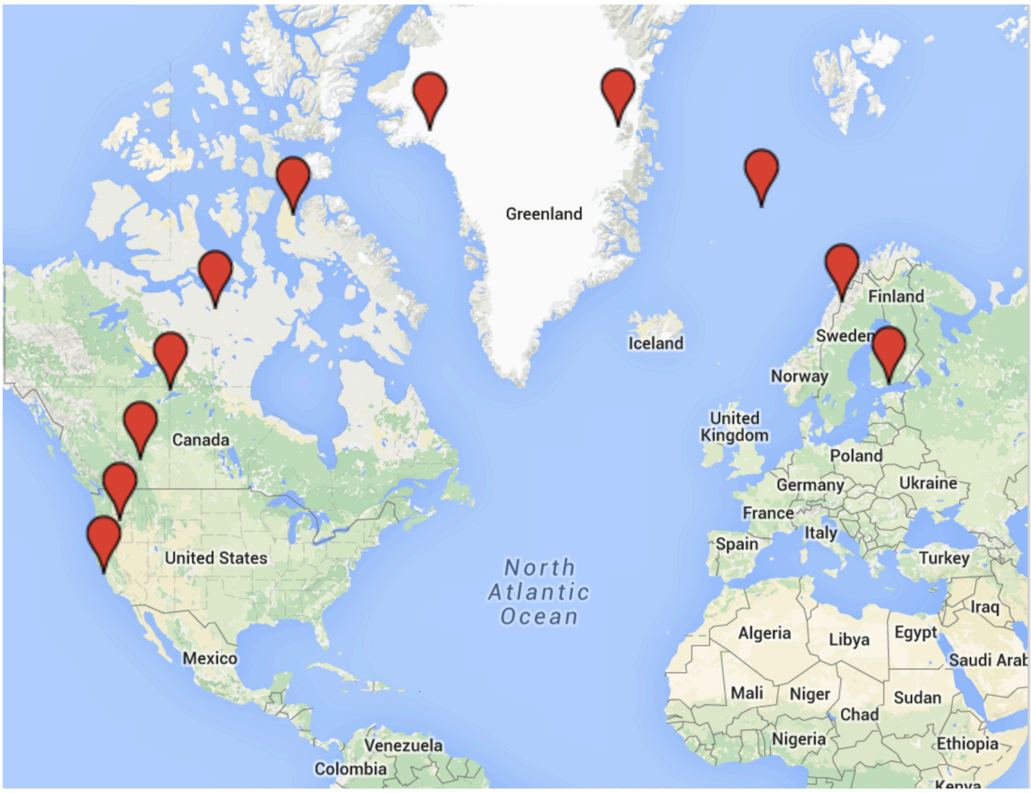
Import the `LengthSpace`

```
import axle.algebra.GeoCoordinates.geoCoordinatesLengthSpace
```

Use it to compute the points at 10% increments from SFO to HEL

```
val midpoints = (0 to 10).map(i => geoCoordinatesLengthSpace.onPath(sfo, hel, i
 / 10d))
```

```
midpoints.map(_.show)
// res25: IndexedSeq[String] = Vector(
//   "37.618900000000004° N 122.37500000000003° W",
//   "45.13070460867812° N 119.34966960499106° W",
//   "52.538395227224065° N 115.40855064022753° W",
//   "59.76229827032038° N 109.88311454897514° W",
//   "66.62843399359917° N 101.39331801935985° W",
//   "72.70253233457194° N 86.91316673834633° W",
//   "76.8357649372965° N 61.093630209243706° W",
//   "77.01752181288721° N 25.892878424459116° W",
//   "73.11964173748505° N -0.9862308621078928° W",
//   "67.1423066577233° N -16.143753987066464° W",
//   "60.3172° N -24.9633° W"
// )
```



Future Work

The methods `over` and `by` are used to multiply and divide other values with units. This behavior is not yet implemented.

- Shapeless for compound Quanta and Bayesian Networks
- Physics (eg, how Volume relates to Flow)
- Rm throws from axle.quanta.UnitConverterGraph

Visualization

See the Gallery for more examples.

Output Formats

The `show` function is available in the `axle._` package. It can be applied to several types of Axle objects.

The package `axle.awt._` contains functions for creating files from the images: `png`, `jpeg`, `gif`, `bmp`.

The package `axle.web._` contains a `svg` function for creating svg files.

For example:

```
vis.show

vis.svg[IO]("filename.svg").unsafeRunSync()

vis.png[IO]("filename.png").unsafeRunSync()
```

Animation

`Plot`, `BarChart`, `BarChartGrouped`, and `ScatterPlot` support animation. The visualizing frame polls for updates at a rate of approximately 24 Hz (every 42 ms).

The `play` command requires the same first argument as `show` does. Additionally, `play` requires a `Observable[D]` function that represents the stream of data updates. The implicit argument is a `monix.execution.Scheduler`.

An `axle.reactive.CurrentValueSubscriber` based on the `Observable[D]` can be used to create the `dataFn` read by the visualization.

See **Bar Chart Animation** for a full example of animation.

Plot

Two-dimensional plots

Example: Plot Random Waves Over Time

Imports

```
import org.joda.time.DateTime

import scala.collection.immutable.TreeMap
import scala.math.sin
```

```
import spire.random.Generator
import spire.random.Generator.rng

import cats.implicits._

import axle._
import axle.visualize._
import axle.joda.dateTimeOrder

import axle.visualize.Color._
```

Generate the time-series to plot

```
val now = new DateTime()

val colors = Vector(red, blue, green, yellow, orange)

def randomTimeSeries(i: Int, gen: Generator) = {
  val φ = gen.nextDouble()
  val A = gen.nextDouble()
  val ω = 0.1 / gen.nextDouble()
  ("series %d %1.2f %1.2f %1.2f".format(i, φ, A, ω),
    new TreeMap[DateTime, Double]() ++
      (0 to 100).map(t => (now.plusMinutes(2 * t) -> A * sin(ω * t +
 φ))).toMap)
}

val waves = (0 until 20).map(i => randomTimeSeries(i, rng)).toList
```

Imports for visualization

```
import cats.Show

import spire.algebra._

import axle.visualize.Plot
import axle.algebra.Plottable.doublePlottable
import axle.joda.dateTimeOrder
import axle.joda.dateTimePlottable
import axle.joda.dateTimeTics
import axle.joda.dateTimeDurationLengthSpace

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
```

Define the visualization

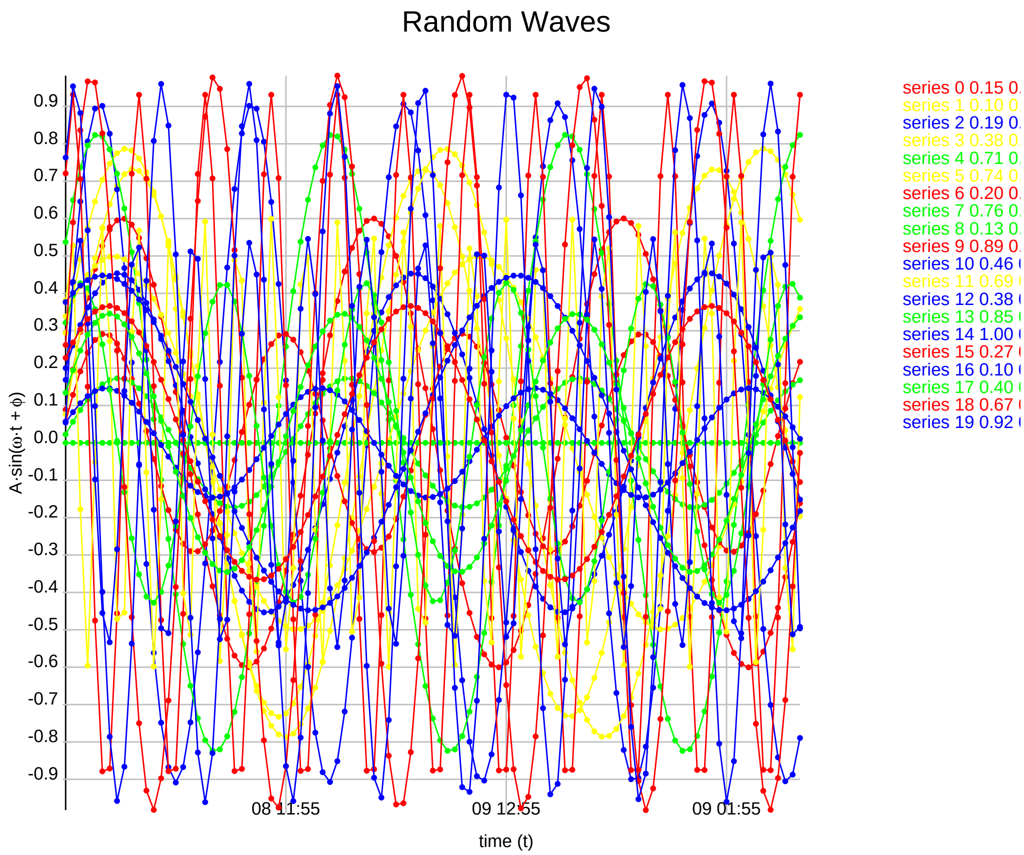
```
val plot = Plot[String, DateTime, Double, TreeMap[DateTime, Double]](
  () => waves,
  connect = true,
  colorOf = s => colors(s.hash.abs % colors.length),
  title = Some("Random Waves"),
  xAxisLabel = Some("time (t)"),
  yAxis = Some(now),
  yAxisLabel = Some("A·sin(ω·t + φ)")).zeroXAxis
```

If instead we had supplied `(Color, String)` pairs, we would have needed something like preciding the `Plot` definition:

```
implicit val showCL: Show[(Color, String)] =
  new Show[(Color, String)] {
    def show(cl: (Color, String)): String = cl._2
  }
// showCL: Show[(Color, String)] = repl.MdocSession$App$$anon$1@138052d4
```

Create the SVG

```
import axle.web._
import cats.effect._

plot.svg[IO]("docwork/images/random_waves.svg").unsafeRunSync()
```



Plot Animation

This example traces two "saw" functions vs time:

Imports

```
import org.joda.time.DateTime
import edu.uci.ics.jung.graph.DirectedSparseGraph
import collection.immutable.TreeMap

import cats.implicits._

import monix.reactive._

import spire.algebra.Field

import axle.jung._
import axle.quanta.Time
import axle.visualize._
import axle.reactive.intervalScan
```

Define stream of data updates refreshing every 500 milliseconds

```
val initialData = List(
  ("saw 1", new TreeMap[DateTime, Double]()),
  ("saw 2", new TreeMap[DateTime, Double]())
)
// initialData: List[(String, TreeMap[DateTime, Double])] = List(
//   ("saw 1", TreeMap()),
//   ("saw 2", TreeMap())
// )

val saw1 = (t: Long) => (t % 10000) / 10000d
// saw1: Long => Double = <function1>
val saw2 = (t: Long) => (t % 100000) / 50000d
// saw2: Long => Double = <function1>

val fs = List(saw1, saw2)
// fs: List[Long => Double] = List(<function1>, <function1>)

val refreshFn = (previous: List[(String, TreeMap[DateTime, Double])]) => {
  val now = new DateTime()
  previous.zip(fs).map({ case (old, f) => (old._1, old._2 ++ Vector(now -
> f(now.getMillis))) })
}
// refreshFn: List[(String, TreeMap[DateTime, Double])] => List[(String,
 TreeMap[DateTime, Double])] = <function1>

implicit val timeConverter = {
  import axle.algebra.modules.doubleRationalModule
  Time.converterGraphK2[Double, DirectedSparseGraph]
}
// timeConverter: quanta.UnitConverterGraph[Time, Double,
 DirectedSparseGraph[quanta.UnitOfMeasurement[Time], Double => Double]] with
 quanta.TimeConverter[Double] = axle.quanta.Time$$anon$1@1fe68c53
import timeConverter.millisecond

val dataUpdates: Observable[Seq[(String, TreeMap[DateTime, Double])]] =
  intervalScan(initialData, refreshFn, 500d *: millisecond)
// dataUpdates: Observable[Seq[(String, TreeMap[DateTime, Double])]] =
 monix.reactive.internal.operators.ScanObservable@730f4817
```

Create `CurrentValueSubscriber`, which will be used by the `Plot` to get the latest values

```
import monix.execution.Scheduler.Implicits.global
import axle.reactive.CurrentValueSubscriber

val cvSub
 = new CurrentValueSubscriber[Seq[(String, TreeMap[DateTime, Double])]]()
val cvCancellable = dataUpdates.subscribe(cvSub)

val plot = Plot[DateTime, Double, TreeMap[DateTime, Double]](
  () => cvSub.currentValue.getOrElse(initialData),
  connect = true,
```

```
  colorOf = (label: String) => Color.black,
  title = Some("Saws"),
  xAxis = Some(0d),
  xAxisLabel = Some("time (t)"),
  yAxisLabel = Some("y")
)
```

Animate

```
import axle.awt._

val (frame, paintCancellable) = play(plot, dataUpdates)
```

Tear down resources

```
paintCancellable.cancel()
cvCancellable.cancel()
frame.dispose()
```

ScatterPlot

```
import axle.visualize._

val data = Map(
  (1, 1) -> 0,
  (2, 2) -> 0,
  (3, 3) -> 0,
  (2, 1) -> 1,
  (3, 2) -> 1,
  (0, 1) -> 2,
  (0, 2) -> 2,
  (1, 3) -> 2)
```

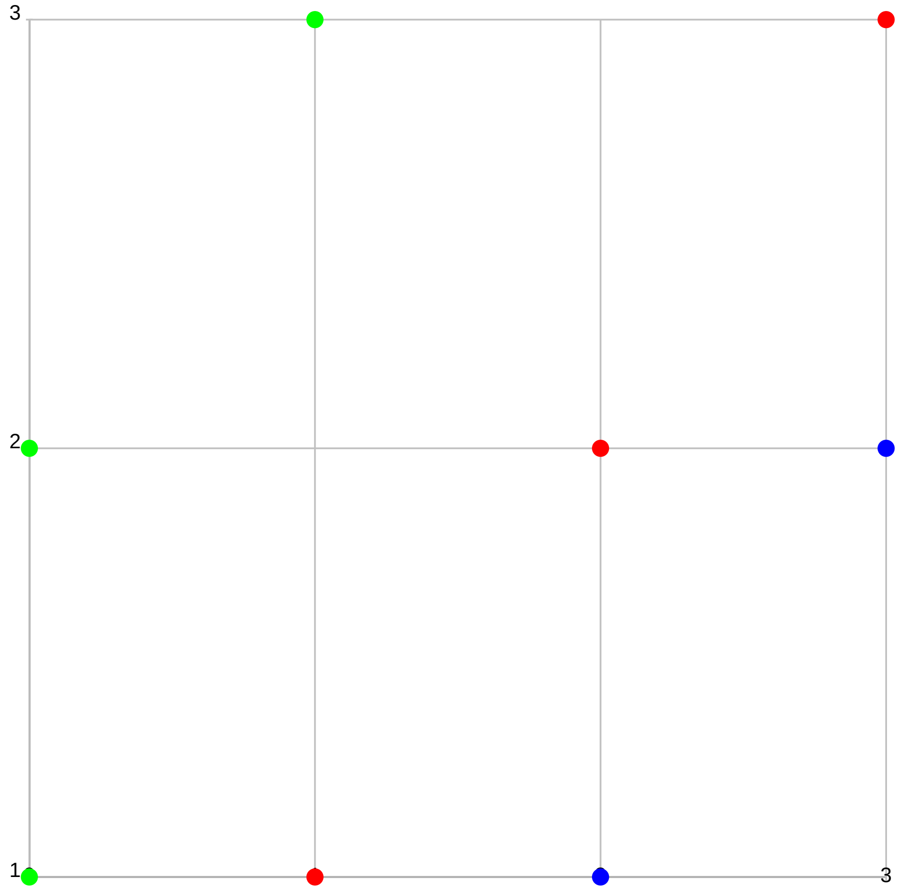
Define the ScatterPlot

```
import axle.visualize.Color._
import cats.implicits._
```

```
val plot = ScatterPlot[String, Int, Int, Map[(Int, Int), Int]](
  () => data,
  colorOf = (x: Int, y: Int) => data((x, y)) match {
    case 0 => red
    case 1 => blue
    case 2 => green
  },
  labelOf = (x: Int, y: Int) => data.get((x, y)).map(s => (s.toString, false)))
```

Create the SVG

```
import axle.web._
import cats.effect._

plot.svg[IO]("docwork/images/scatter.svg").unsafeRunSync()
```



BarChart

Two-dimensional bar charts.

Example: Fruit Sales BarChart

The dataset:

```
val sales = Map(
  "apple" -> 83.8,
  "banana" -> 77.9,
  "coconut" -> 10.1
)
```

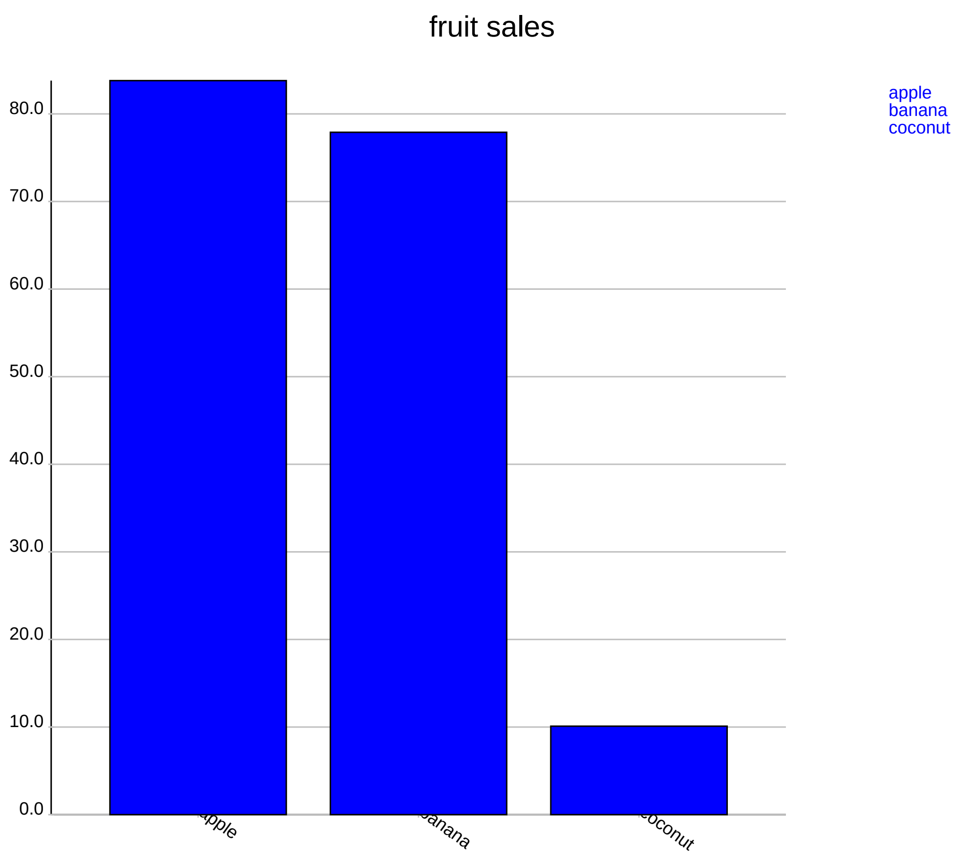
Define a bar chart visualization

```
import spire.algebra.Field
import cats.implicits._
import axle.visualize.BarChart
import axle.visualize.Color.lightGray
```

```
implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra

val chart = BarChart[String, Double, Map[String, Double], String](
  () => sales,
  title = Some("fruit sales"),
  hoverOf = (c: String) => Some(c),
  linkOf = (c: String) => Some((new java.net.URL(s"http://wikipedia.org/
wiki/$c"), lightGray))
)
```

Create the SVG

```
import axle.web._
import cats.effect._

chart.svg[IO]("docwork/images/fruitsales.svg").unsafeRunSync()
```



BarChart Animation

This example keeps the "bar" value steady at 1.0 while assigning a new random Double (between 0 and 1) to "foo" every second.

Imports

```
import scala.util.Random.nextDouble
import axle.jung._
import axle.quanta.Time
import edu.uci.ics.jung.graph.DirectedSparseGraph
import monix.reactive._
import axle.reactive.intervalScan
```

Define stream of data updates

```
val groups = Vector("foo", "bar")
val initial = Map("foo" -> 1d, "bar" -> 1d)

val tick = (previous: Map[String, Double]) => previous + ("foo" -
> nextDouble())

implicit val timeConverter = {
  import axle.algebra.modules.doubleRationalModule
  Time.converterGraphK2[Double, DirectedSparseGraph]
}
import timeConverter.second

val dataUpdates: Observable[Map[String, Double]]
 = intervalScan(initial, tick, 1d *: second)
```

Create `CurrentValueSubscriber`, which will be used by the `BarChart` to get the latest value

```
import axle.reactive.CurrentValueSubscriber
import monix.execution.Scheduler.Implicits.global

val cvSub = new CurrentValueSubscriber[Map[String, Double]]()
val cvCancellable = dataUpdates.subscribe(cvSub)

import axle.visualize.BarChart

val chart = BarChart[String, Double, Map[String, Double], String](
  () => cvSub.currentValue.getOrElse(initial),
  title = Some("random")
)
```

Animate

```
import axle.awt.play

val (frame, paintCancellable) = play(chart, dataUpdates)
```

Tear down the resources

```
paintCancellable.cancel()
cvCancellable.cancel()
frame.dispose()
```

GroupedBarChart

Two-dimensional grouped bar charts

Example: Fruit Sales Grouped By Year

The following example dataset:

```
val sales = Map(
  ("apple", 2011) -> 43.0,
  ("apple", 2012) -> 83.8,
  ("banana", 2011) -> 11.3,
  ("banana", 2012) -> 77.9,
  ("coconut", 2011) -> 88.0,
  ("coconut", 2012) -> 10.1
)
```

Shared imports

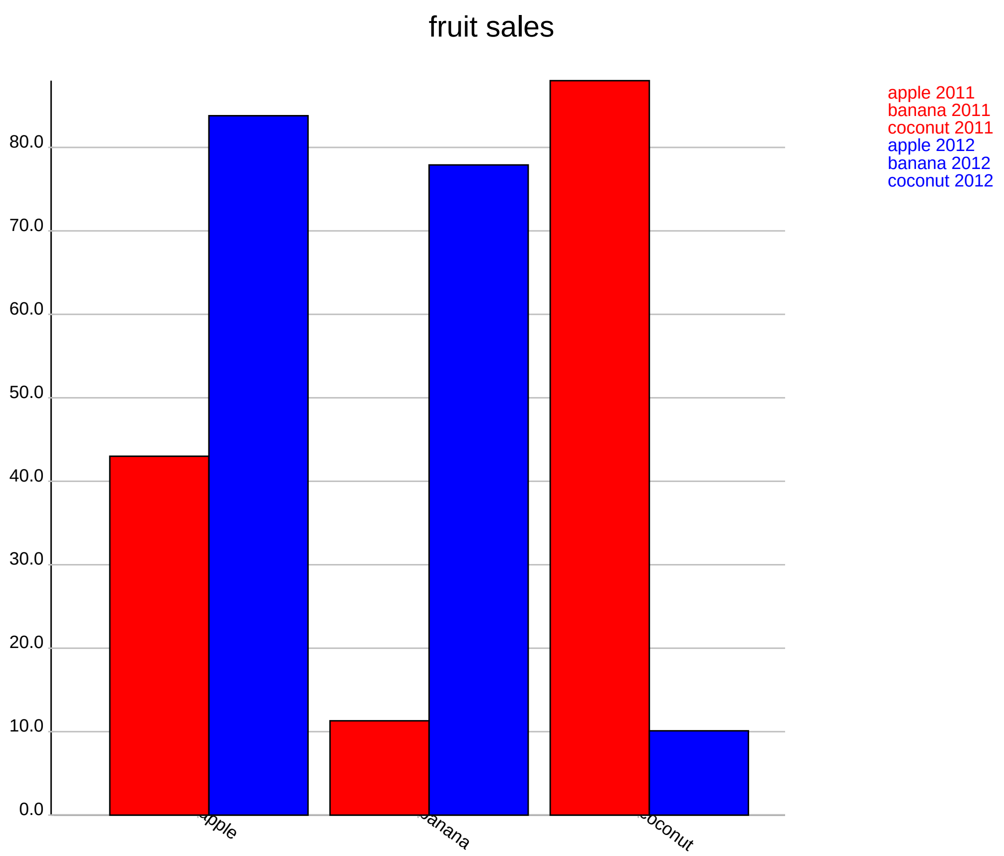
```
import cats.implicits._
import spire.algebra.Field
import axle.visualize.BarChartGrouped
import axle.visualize.Color._

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
```

The data can be grouped in two ways to produce bar charts:

```
val chart
 = BarChartGrouped[String, Int, Double, Map[(String, Int), Double], String](
  () => sales,
  title = Some("fruit sales"),
  colorOf = (label: String, year: Int) => year match {
    case 2011 => red
    case 2012 => blue
  }
)
```

Create the SVG

```
import axle.web._
import cats.effect._

chart.svg[IO]("docwork/images/barchart1.svg").unsafeRunSync()
```

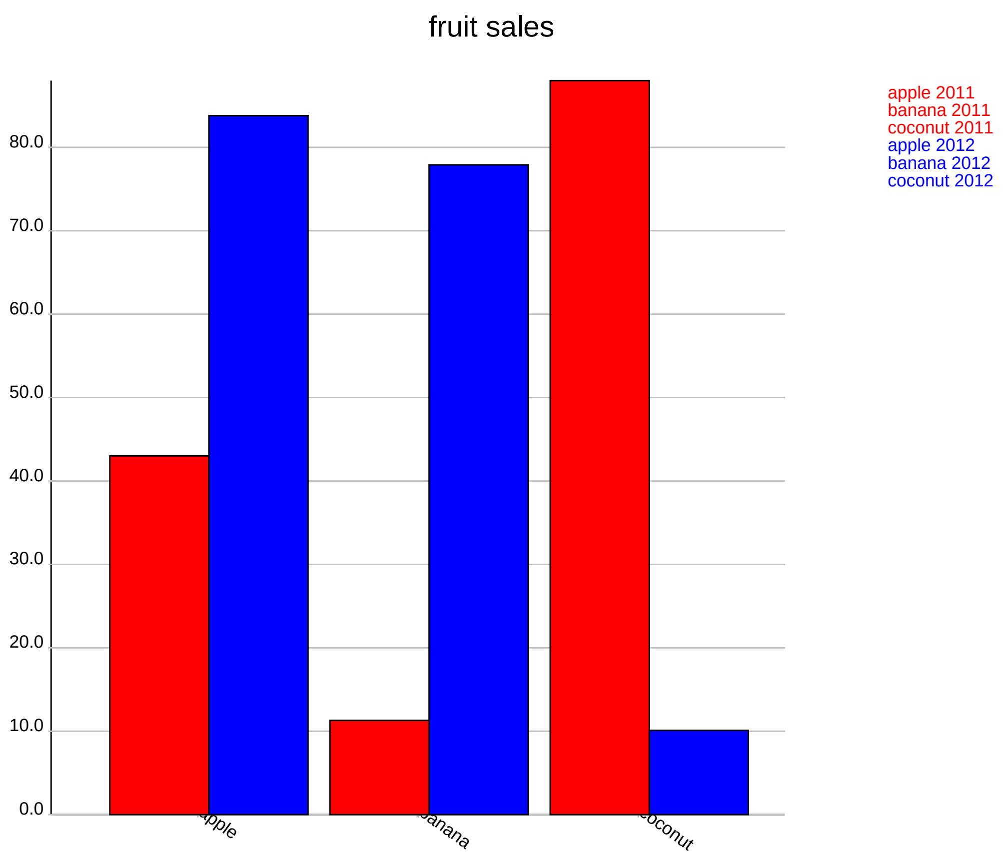


Or alternatively

```
val chart2
 = BarChartGrouped[Int, String, Double, Map[(Int, String), Double], String](
  () => sales map { case (k, v) => (k._2, k._1) -> v},
  colorOf = (year: Int, label: String) => label match {
    case "apple" => red
    case "banana" => yellow
    case "coconut" => brown
  },
  title = Some("fruit sales")
)
```

Create the second SVG

```
import axle.web._
import cats.effect._

chart.svg[IO]("docwork/images/barchart2.svg").unsafeRunSync()
```



Pixelated Colored Area

This visualization shows the composition of a function `f: (X, Y) => V` with a colorizing function `c: V => Color` over a rectangular range on the `(X, Y)` plane. `LengthSpace[X, X, Double]` and `LengthSpace[Y, Y, Double]` must be implicitly in scope.

Example: Red to Yellow Diagonal

A few imports:

```
import cats.implicits._

import axle._
import axle.visualize._
```

Define a function to compute an `Double` for each point on the plane `(x, y): (Double, Double)`

```
def f(x0: Double, x1: Double, y0: Double, y1: Double) =
  x0 + y0
```

Define a `toColor` function. Here we first prepare an array of colors to avoid creating the objects during rendering.

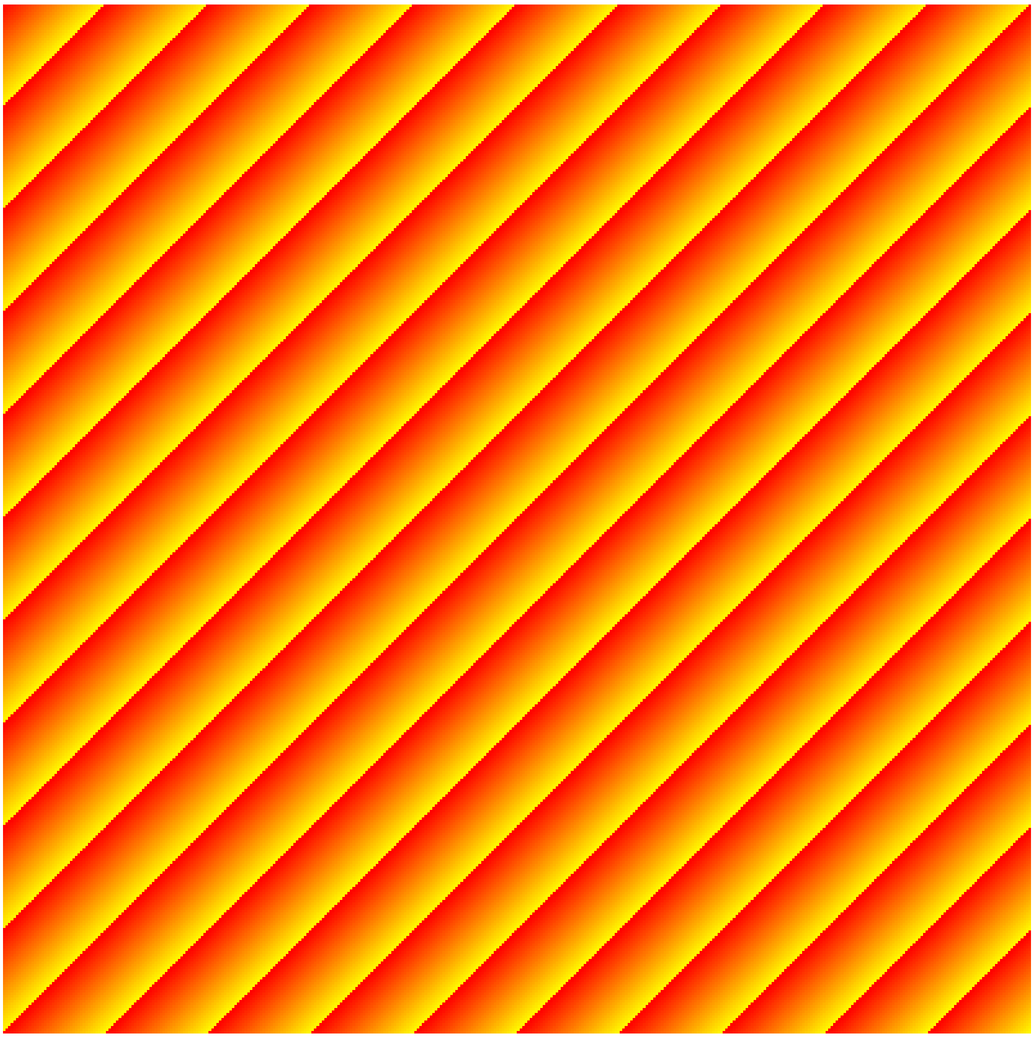
```
val n = 100

// red to orange to yellow
val roy = (0 until n).map( i =>
  Color(255, ((i / n.toDouble) * 255).toInt, 0)
).toArray

def toColor(v: Double) = roy(v.toInt % n)
```

Define a `PixelatedColoredArea` to show `toColor # f` over the range `(0,0)` to `(1000,1000)` represented as a 400 pixel square.

```
val pca = PixelatedColoredArea(f, toColor, 400, 400, 0d, 1000d, 0d, 1000d)
```

Create PNG

```
import axle.awt._
import cats.effect._

pca.png[IO]("docwork/images/roy_diagonal.png").unsafeRunSync()
```



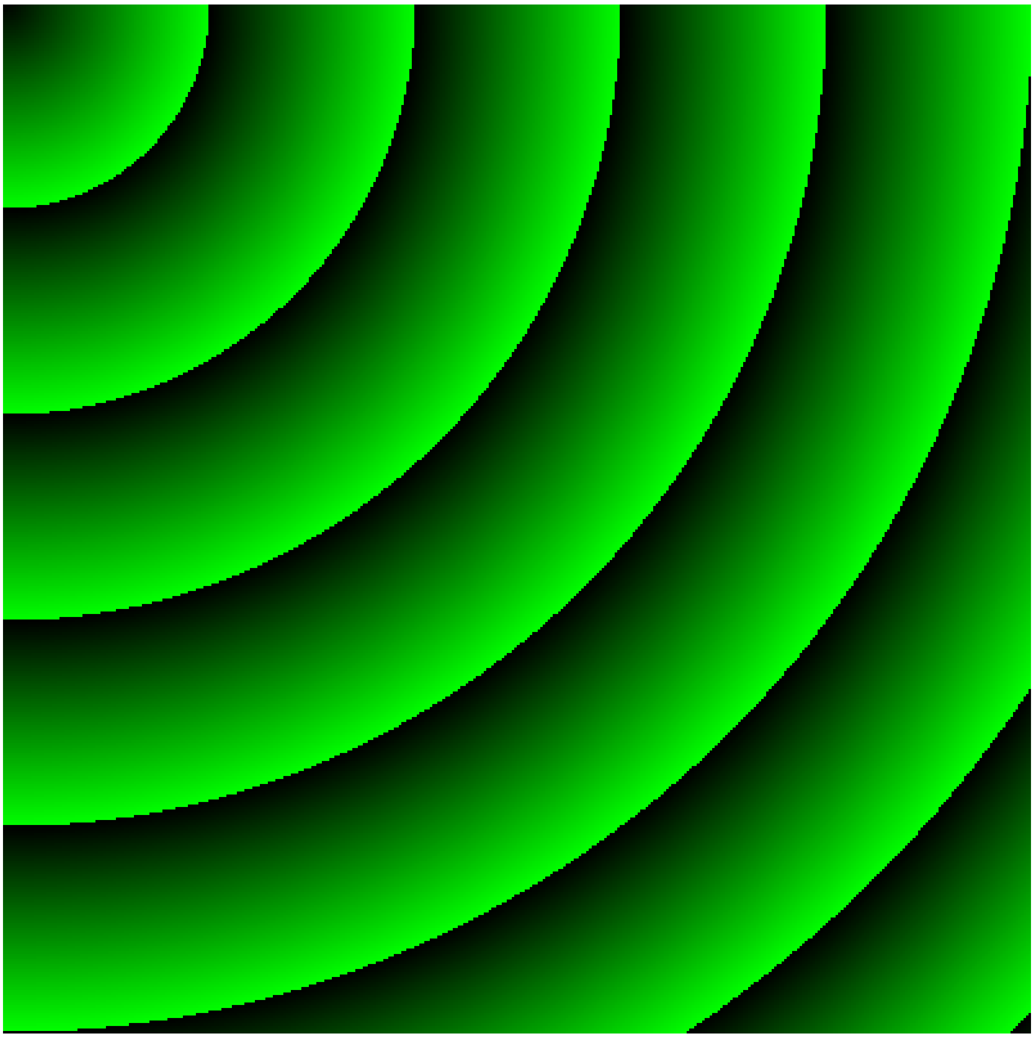
Example: Green Polar

More compactly:

```
import spire.math.sqrt

val m = 200

val greens = (0 until m).map( i =>
  Color(0, ((i / m.toDouble) * 255).toInt, 0)
).toArray

val gpPca = PixelatedColoredArea(
    (x0: Double, x1: Double, y0: Double, y1: Double) => sqrt(x0*x0 + y0*y0),
    (v: Double) => greens(v.toInt % m),
    400, 400,
    0d, 1000d,
    0d, 1000d)
```

Create the PNG

```
import axle.awt._
import cats.effect._

gpPca.png[IO]("docwork/images/green_polar.png").unsafeRunSync()
```



Future Work

- WebGL
- SVG Animation
- Box Plot
- Candlestick Chart
- Honor graph vis params in awt graph visualizations
- `axle.web.Table` and `HtmlFrom[Table[T]]`

- Log scale
- `SVG[Matrix]`
- `BarChart` Variable width bars
- Horizontal barchart
- `KMeansVisualization` / `ScatterPlot` similarity (at least DataPoints)
- `SVG[H]` for BarChart hover (wrap with \<g> to do getBBox)
- Background box for `ScatterPlot` hover text?
- Fix multi-color cube rendering
- Bloom filter surface
- Factor similarity between SVG and Draw?
- Re-enable `axle-jogl`
- May require jogamop 2.4, which is not yet released
- Or possibly use **jogamp archive**
- See processing's approach in **this commit**
- Unchecked constraint in PlotDataView

Graph Theory

Currently implemented for the Jung library.

Eventually Axle will provide its own basic Graph implementation, which will remove Jung as a dependency of `axle-core`.

Directed Graph

Example with `String` is the vertex value and an `Edge` type with two values (a `String` and an `Int`) to represent the edges

```
val (a, b, c, d) = ("a", "b", "c", "d")

class Edge(val s: String, val i: Int)
```

Invoke the `DirectedGraph` typeclass with type parameters that denote that we will use Jung's `DirectedSparseGraph` as the graph type, with `String` and `Edge` as vertex and edge values, respectively.

```
import edu.uci.ics.jung.graph.DirectedSparseGraph
import axle.algebra._
import axle.jung._

val jdg = DirectedGraph.k2[DirectedSparseGraph, String, Edge]
```

Use the `jdg` witness's `make` method to create the directed graph

```
val dg = jdg.make(List(a, b, c, d),
  List(
    (a, b, new Edge("hello", 1)),
    (b, c, new Edge("world", 4)),
    (c, d, new Edge("hi", 3)),
    (d, a, new Edge("earth", 1)),
    (a, c, new Edge("!", 7)),
    (b, d, new Edge("hey", 2))))
```

```
import cats.implicits._
import axle.syntax.directedgraph.directedGraphOps
import axle.syntax.finite.finiteOps

dg.vertexProjection.size
// res1: Int = 4

dg.edgeProjection.size
// res2: Int = 6
```

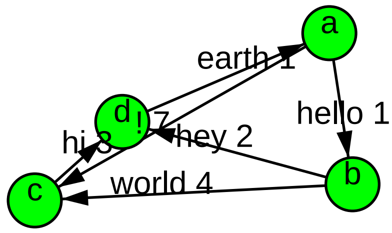
```
dg.findVertex(_ === "a").map(v => dg.successors(v))
// res3: Option[Set[String]] = Some(value = Set("b", "c"))

dg.findVertex(_ === "c").map(v => dg.successors(v))
// res4: Option[Set[String]] = Some(value = Set("d"))

dg.findVertex(_ === "c").map(v => dg.predecessors(v))
// res5: Option[Set[String]] = Some(value = Set("a", "b"))

dg.findVertex(_ === "c").map(v => dg.neighbors(v))
// res6: Option[Set[String]] = Some(value = Set("a", "b", "d"))
```

Create a Visualization of the graph

```
import cats.Show

implicit val showEdge: Show[Edge] = new Show[Edge] {
  def show(e: Edge): String = e.s + " " + e.i
}

import axle.visualize._

val dVis
 = DirectedGraphVisualization[DirectedSparseGraph[String, Edge], String, Edge](
  dg,
  width = 300,
  height = 300,
  border = 10,
  radius = 10,
  arrowLength = 10,
  color = Color.green,
  borderColor = Color.black,
  fontSize = 12
)
```

Render as sn SVG file

```
import axle.web._
import cats.effect._

dVis.svg[IO]("docwork/images/SimpleDirectedGraph.svg").unsafeRunSync()
```



Undirected Graph

An undirected graph using the same dataa:

```
val (a, b, c, d) = ("a", "b", "c", "d")

class Edge(val s: String, val i: Int)
```

Invoke the `UndirectedGraph` typeclass with type parameters that denote that we will use Jung's `UndirectedSparseGraph` as the graph type, with `String` and `Edge` as vertex and edge values, respectively.

```
import edu.uci.ics.jung.graph.UndirectedSparseGraph
import axle.algebra._
import axle.jung._

val jug = UndirectedGraph.k2[UndirectedSparseGraph, String, Edge]
```

Use the `jug` witness's `make` method to create the undirected graph

```
val ug = jug.make(List(a, b, c, d),
  List(
    (a, b, new Edge("hello", 10)),
    (b, c, new Edge("world", 1)),
    (c, d, new Edge("hi", 3)),
    (d, a, new Edge("earth", 7)),
    (a, c, new Edge("!", 1)),
    (b, d, new Edge("hey", 2))))
```

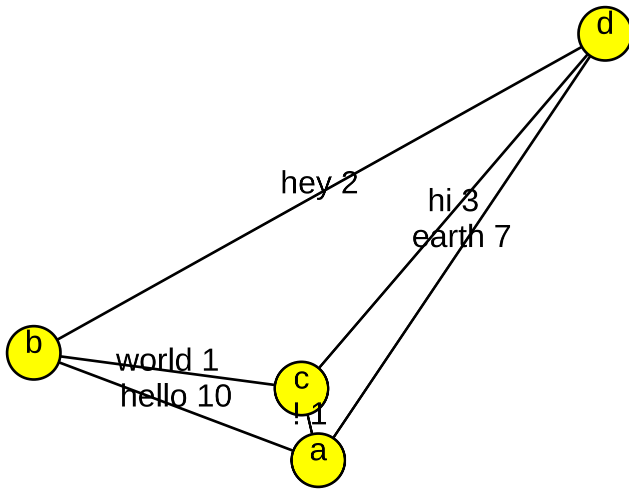
```
import cats.implicits._
import axle.syntax.undirectedgraph.undirectedGraphOps
import axle.syntax.finite.finiteOps

ug.vertexProjection.size
// res9: Int = 4

ug.edgeProjection.size
// res10: Int = 6

ug.findVertex(_ == "c").map(v => ug.neighbors(v))
// res11: Option[Iterable[String]] = Some(value = Iterable("a", "b", "d"))

ug.findVertex(_ == "a").map(v => ug.neighbors(v))
// res12: Option[Iterable[String]] = Some(value = Iterable("b", "c", "d"))
```

Create a Visualization of the graph

```
import cats.Show

implicit val showEdge: Show[Edge] = new Show[Edge] {
  def show(e: Edge): String = e.s + " " + e.i
}

import axle.visualize._

val uVis
 = UndirectedGraphVisualization[UndirectedSparseGraph[String, Edge], String, Edge]
(
  ug,
  width = 300,
  height = 300,
  border = 10,
  color = Color.yellow)
```

Render as an SVG file

```
import axle.web._
import cats.effect._

uVis.svg[IO]("docwork/images/SimpleUndirectedGraph.svg").unsafeRunSync()
```



Randomness and Uncertainty

Probability Model

Modeling probability, randomness, and uncertainly is one of the primary objectives of Axle.

The capabilities are available via four typeclasses and one trait

- Sampler
- Region (trait modeling #-algebra)
- Kolmogorov
- Bayes
- Monad (`cats.Monad`)

Concrete number type are avoided in favor of structures from Abstract Algebra -- primarily `Ring` and `Field`. These are represented as context bounds, usually passed implicitly.

The examples in this document use the `spire.math.Rational` number type, but work as well for `Double`, `Float`, etc. The precise number type `Rational` is used in tests because their precision allows the assertions to be expressed without any error tolerance.

Imports

Preamble to pull in the commonly-used functions in this section:

```
import cats.implicits._
import cats.effect._

import spire.math._
import spire.algebra._

import axle.probability._
import axle.algebra._
import axle.visualize._
import axle.web._
```

Creating Probability Models

There are a few type of probability models in Axle. The simplest is the `ConditionalProbabilityTable`, which is used throughout this document.

`axle.data.Coin.flipModel` demonstrates a very simple probability model for type `Symbol`.

This is its implementation:

```
val head = Symbol("HEAD")
val tail = Symbol("TAIL")

def flipModel(pHead: Rational
 = Rational(1, 2)): ConditionalProbabilityTable[Symbol, Rational] =
  ConditionalProbabilityTable[Symbol, Rational](
    Map(
      head -> pHead,
      tail -> (1 - pHead)))
```

Its argument is the bias for the `HEAD` side. Without a provided bias, it is assumed to be a fair coin.

```
val fairCoin = flipModel()
// fairCoin: ConditionalProbabilityTable[Symbol, Rational] =
 ConditionalProbabilityTable(
//   p = Map('HEAD -> 1/2, 'TAIL -> 1/2)
// )

val biasedCoin = flipModel(Rational(9, 10))
// biasedCoin: ConditionalProbabilityTable[Symbol, Rational] =
 ConditionalProbabilityTable(
//   p = Map('HEAD -> 9/10, 'TAIL -> 1/10)
// )
```

Rolls of dice are another common example.

```
def rollModel(n: Int): ConditionalProbabilityTable[Int, Rational] =
  ConditionalProbabilityTable(
    (1 to n).map(i => (i, Rational(1, n.toLong))).toMap)
```

The values `d6` and `d10` model rolls of 6 and 10-sided dice.

```
val d6 = rollModel(6)
// d6: ConditionalProbabilityTable[Int, Rational] =
 ConditionalProbabilityTable(
//   p = HashMap(5 -> 1/6, 1 -> 1/6, 6 -> 1/6, 2 -> 1/6, 3 -> 1/6, 4 -> 1/6)
// )

val d10 = rollModel(10)
// d10: ConditionalProbabilityTable[Int, Rational] =
 ConditionalProbabilityTable(
//   p = HashMap(
//     5 -> 1/10,
//     10 -> 1/10,
//     1 -> 1/10,
//     6 -> 1/10,
//     9 -> 1/10,
//     2 -> 1/10,
//     7 -> 1/10,
```

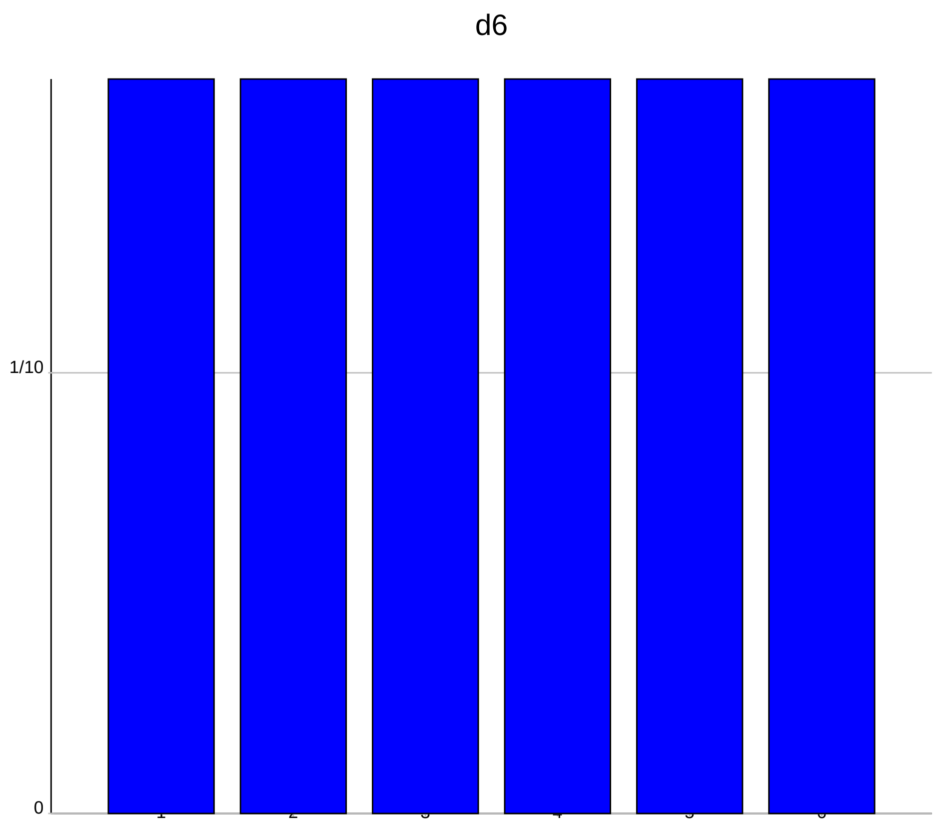
```
//      3 -> 1/10,
//      8 -> 1/10,
//      4 -> 1/10
//    )
// )
```

Define a visualization of the distribution of events in the **d6** model:

```
val d6vis
 = BarChart[Int, Rational, ConditionalProbabilityTable[Int, Rational], String](
  () => d6,
  colorOf = _ => Color.blue,
  xAxis = Some(Rational(0)),
  title = Some("d6"),
  labelAngle = Some(0d *: angleDouble.degree),
  drawKey = false)
```

Create an SVG

```
d6vis.svg[IO]("docwork/images/d6.svg").unsafeRunSync()
```



Sampler

The `Sampler` typeclass provides the ability to "execute" the model and product a random sample via the `sample` method.

It's type signature is:

```
def sample(gen: Generator)
(implicit spireDist: Dist[V], ringV: Ring[V], orderV: Order[V]): A
```

These imports make available a `Generator` as source of entropy

```
import spire.random._

val rng = Random.generatorFromSeed(Seed(42))
// rng: spire.random.rng.Cmwc5 = spire.random.rng.Cmwc5@7c47a58f
```

And then the `.sample` syntax:

```
import axle.syntax.sampler._
```

`sample` requires a Spire `Generator`. It also requires context bounds on the value type `V` that give the method the ability to produces values with a distribution conforming to the probability model.

```
(1 to 10) map { _ => fairCoin.sample(rng) }
// res1: IndexedSeq[Symbol] = Vector(
//   'HEAD,
//   'HEAD,
//   'HEAD,
//   'TAIL,
//   'TAIL,
//   'HEAD,
//   'TAIL,
//   'TAIL,
//   'TAIL,
//   'HEAD
// )
```

```
(1 to 10) map { _ => biasedCoin.sample(rng) }
// res2: IndexedSeq[Symbol] = Vector(
//   'HEAD,
//   'HEAD,
//   'HEAD,
//   'HEAD,
//   'HEAD,
//   'HEAD,
//   'HEAD,
//   'HEAD,
```

```
//   'HEAD,
//   'HEAD
// )
```

```
(1 to 10) map { _ => d6.sample(rng) }
// res3: IndexedSeq[Int] = Vector(3, 2, 5, 2, 6, 2, 6, 1, 1, 5)
```

Simulate 1k rolls of one d6

```
val rolls = (0 until 1000) map { _ => d6.sample(rng) }
```

Then tally them

```
implicit val ringInt: CRing[Int] = spire.implicits.IntAlgebra

import axle.syntax.talliable._
```

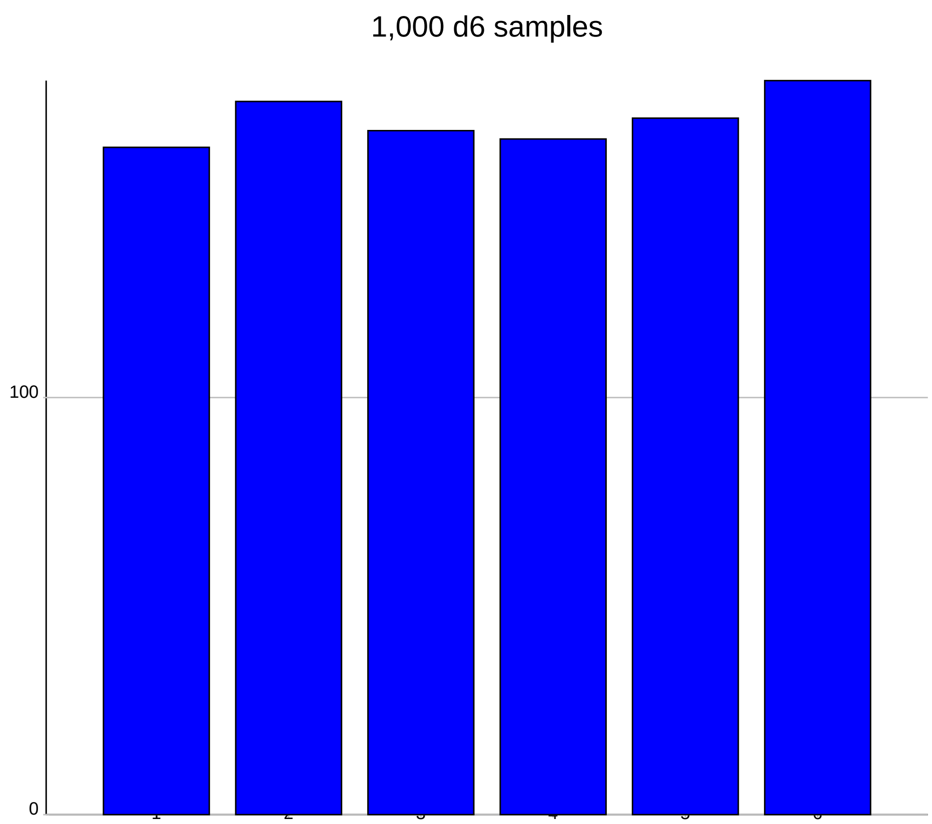
```
val oneKd6Histogram = rolls.tally
// oneKd6Histogram: Map[Int, Int] = Map(
//   5 -> 167,
//   1 -> 160,
//   6 -> 176,
//   2 -> 171,
//   3 -> 164,
//   4 -> 162
// )
```

Create a visualization

```
val d6oneKvis = BarChart[Int, Int, Map[Int, Int], String](
  () => oneKd6Histogram,
  colorOf = _ => Color.blue,
  xAxis = Some(0),
  title = Some("1,000 d6 samples"),
  labelAngle = Some(0d *: angleDouble.degree),
  drawKey = false)
```

Create the SVG

```
d6oneKvis.svg[IO]("docwork/images/d6-1Ksamples.svg").unsafeRunSync()
```



Sigma Algebra Regions

The sealed `Region[A]` trait is extended by the following case classes that form a way to describe expressions on the event-space of a probability model. In Measure Theory, these expressions are said to form a "sigma-algebra" ("#-algebra")

In order of arity, they case classes extending this trait are:

Arity 0

- `RegionEmpty` never matches any events or probability mass
- `RegionAll` always matches all events and probability mass

Arity 1 (not including typeclass witnesses)

- `RegionEq` matches when an event is equal to the supplied object, with respect to the supplied `cats.kernel.Eq[A]` witness.
- `RegionIf` matches when the supplied condition function returns `true`
- `RegionSet` matches when an event is contained within the supplied `Set[A]`.
- `RegionNegate` negates the supplied region.

- `RegionGTE` is true when an event is greater than or equal to the supplied object, with respect to the supplied `cats.kernel.Order[A]`
- `RegionLTE` is true when an event is less than or equal to the supplied object, with respect to the supplied `cats.kernel.Order[A]`

Arity 2

- `RegionAnd` is the conjunction of both arguments. It can be created using the `and` method in the `Region` trait.
- `RegionOr` is the disjunction of both arguments. It can be created using the `or` method in the `Region` trait.

Note that a "Random Variable" does not appear in this discussion. The `axle.probability.Variable` class does define a `is` method that returns a `RegionEq`, but the probability models themselves are not concerned with the notion of a `Variable`. They are simply models over regions of events on their single, opaque type that adhere to the laws of probability.

The eventual formalization of `Region` should connect it with a #-algebra from Meaasure Theory.

Kolmogorov for querying Probability Models

probabilityOf (aka "P")

The method `probabilityOf` is defined in terms of a `Region`.

```
def probabilityOf(predicate: Region[A])(implicit fieldV: Field[V]): V
```

Note that `probabilityOf` is aliased to `P` in `axle.syntax.kolmogorov._`

```
import axle.syntax.kolmogorov._
```

The probability of a `head` for a single toss of a fair coin is `1/2`

```
fairCoin.P(RegionEq(head))
// res5: Rational = 1/2
```

The probability that a toss is not `head` is also `1/2`.

```
fairCoin.P(RegionNegate(RegionEq(head)))
// res6: Rational = 1/2
```

The probability that a toss is both `head` and `tail` is zero.

```
fairCoin.P(RegionEq(head) and RegionEq(tail))
// res7: Rational = 0
```

The probability that a toss is either `head` or `tail` is one.

```
fairCoin.P(RegionEq(head) or RegionEq(tail))
// res8: Rational = 1
```

Kolmogorov's Axioms

The single `probabilityOf` method together with the `Region` trait is enough to define Kolmogorov's Axioms of Probability. The axioms are implemented in `axle.laws.KolmogorovProbabilityAxioms` and checked during testing with ScalaCheck.

Basic Measure

Probabilities are non-negative.

```
model.P(region) >= Field[V].zero
```

Unit Measure

The sum the probabilities of all possible events is `one`

```
model.P(RegionAll()) === Field[V].one
```

Combination

For disjoint event regions, `e1` and `e2`, the probability of their disjunction `e1 or e2` is equal to the sum of their independent probabilities.

```
(!((e1 and e2) === RegionEmpty() )) || (model.P(e1 or e2) === model.P(e1)
 + model.P(e2))
```

Bayes Theorem, Conditioning, and Filtering

The `Bayes` typeclass implements the conditioning of a probability model via the `filter` (`|` is also an alias).

```
def filter(predicate: Region[A])(implicit fieldV: Field[V]): M[A, V]
```

Syntax is available via this import

```
import axle.syntax.bayes._
```

`filter` -- along with `probabilityOf` from `Kolomogorov` -- allows Bayes' Theorem to be expressed and checked with ScalaCheck.

```
model.filter(b).P(a) * model.P(b) === model.filter(a).P(b) * model.P(a)
```

For non-zero `model.P(a)` and `model.P(b)`

The theorem is more recognizable as `P(A|B) = P(B|A) * P(A) / P(B)`

Filter is easier to motivate with composite types, but two examples with a d6 show the expected semantics:

Filtering the d6 roll model to 1 and 5:

```
d6.filter(RegionIf(_ % 4 == 1))
// res9: ConditionalProbabilityTable[Int, Rational] =
 ConditionalProbabilityTable(
//   p = Map(5 -> 1/2, 1 -> 1/2)
// )
```

Filter the d6 roll model to 1, 2, and 3:

```
d6.filter(RegionLTE(3))
// res10: ConditionalProbabilityTable[Int, Rational] =
 ConditionalProbabilityTable(
//   p = Map(1 -> 1/3, 2 -> 1/3, 3 -> 1/3)
// )
```

Probability Model as Monads

The `pure`, `map`, and `flatMap` methods of `cats.Monad` are defined for `ConditionalProbabilityTable`, `TallyDistribution`.

Monad Laws

The short version is that the three methods are constrained by a few laws that make them very useful for composing programs. Those laws are:

- Left identity: `pure(x).flatMap(f) === f(x)`
- Right identity: `model.flatMap(pure) === model`
- Associativity: `model.flatMap(f).flatMap(g) === model.flatMap(f.flatMap(g))`

Monad Syntax

There is syntax support in `cats.implicits._` for all three methods.

However, due to limitations of Scala's type inference, it cannot see `ConditionalProbabilityTable[E, V]` as the `M[_]` expected by `Monad`.

The most straigtfoward workaround is just to conjure the monad witness directly and use it, passing the model in as the sole argument to the first parameter group.

```
val monad = ConditionalProbabilityTable.monadWitness[Rational]
```

```
monad.flatMap(d6) { a => monad.map(d6) { b => a + b } }
// res11: ConditionalProbabilityTable[Int, Rational] =
 ConditionalProbabilityTable(
//   p = HashMap(
//     5 -> 1/9,
//     10 -> 1/12,
//     6 -> 5/36,
//     9 -> 1/9,
//     2 -> 1/36,
//     12 -> 1/36,
//     7 -> 1/6,
//     3 -> 1/18,
//     11 -> 1/18,
//     8 -> 5/36,
//     4 -> 1/12
//   )
// )
```

Another strategy to use `map` and `flatMap` directly on the model is a type that can be seen as `M[_]` along with a type annotation:

```
type CPTR[E] = ConditionalProbabilityTable[E, Rational]

(d6: CPTR[Int]).flatMap { a => (d6: CPTR[Int]).map { b => a + b } }
```

Or similar to use a for comprehension:

```
for {
  a <- d6: CPTR[Int]
  b <- d6: CPTR[Int]
} yield a + b
```

Chaining models

Chain two events' models

```
val bothCoinsModel = monad.flatMap(fairCoin) { flip1 =>
  monad.map(fairCoin) { flip2 =>
    (flip1, flip2)
```

```
  }
}
// bothCoinsModel: ConditionalProbabilityTable[(Symbol, Symbol), Rational] =
 ConditionalProbabilityTable(
//   p = HashMap(
//     ('HEAD, 'HEAD) -> 1/4,
//     ('TAIL, 'HEAD) -> 1/4,
//     ('TAIL, 'TAIL) -> 1/4,
//     ('HEAD, 'TAIL) -> 1/4
//   )
// )
```

This creates a model on events of type `(Symbol, Symbol)`

It can be queried with `P` using `RegionIf` to check fields within the `Tuple2`.

```
type TWOFLIPS = (Symbol, Symbol)

bothCoinsModel.P(RegionIf[TWOFLIPS](_._1 == head) and RegionIf[TWOFLIPS](_._2
 == head))
// res14: Rational = 1/4

bothCoinsModel.P(RegionIf[TWOFLIPS](_._1 == head) or RegionIf[TWOFLIPS](_._2
 == head))
// res15: Rational = 3/4
```

Summing two dice rolls

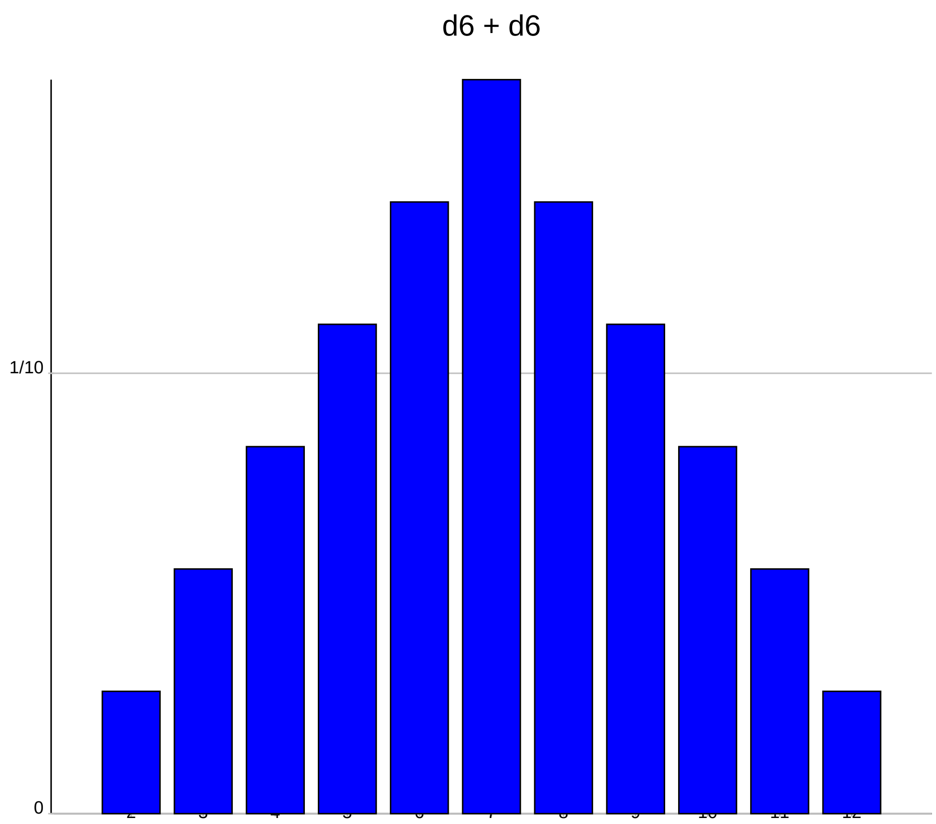
```
val twoDiceSummed = monad.flatMap(d6) { a =>
  monad.map(d6) { b =>
    a + b
  }
}
// twoDiceSummed: ConditionalProbabilityTable[Int, Rational] =
 ConditionalProbabilityTable(
//   p = HashMap(
//     5 -> 1/9,
//     10 -> 1/12,
//     6 -> 5/36,
//     9 -> 1/9,
//     2 -> 1/36,
//     12 -> 1/36,
//     7 -> 1/6,
//     3 -> 1/18,
//     11 -> 1/18,
//     8 -> 5/36,
//     4 -> 1/12
//   )
// )
```

Create a visualization

```
val monadicChart
 = BarChart[Int, Rational, ConditionalProbabilityTable[Int, Rational], String](
  () => twoDiceSummed,
  colorOf = _ => Color.blue,
  xAxis = Some(Rational(0)),
  title = Some("d6 + d6"),
  labelAngle = Some(0d *: angleDouble.degree),
  drawKey = false)
```

Create SVG

```
monadicChart.svg[IO]("docwork/images/distributionMonad.svg").unsafeRunSync()
```



Iffy

A stochastic version of `if` (aka `iffy`) can be implemented in terms of `flatMap` using this pattern for any probability model type `M[A]` such that a `Monad` is defined.

```
def iffy[A, B, M[_]: Monad](
  input      : M[A],
  predicate  : A => Boolean,
  trueClause : M[B],
  falseClause: M[B]): M[B] =
  input.flatMap { i =>
    if( predicate(i) ) {
      trueClause
    } else {
      falseClause
    }
  }
```

An example of that pattern: "if heads, d6+d6, otherwise d10+d10"

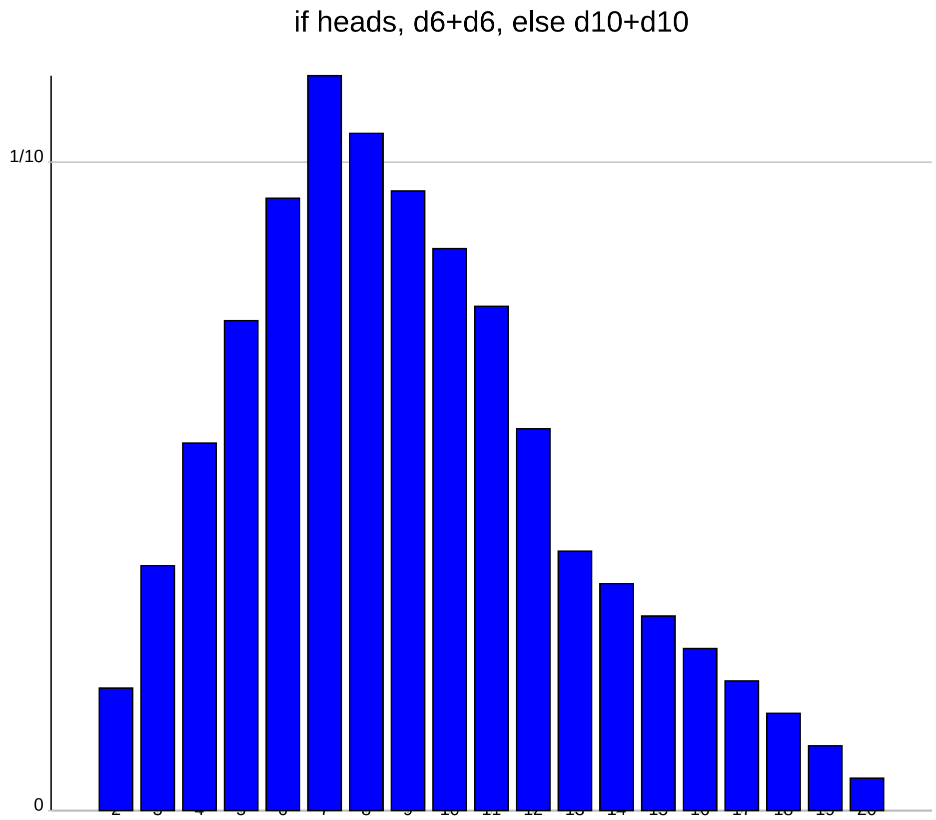
```
import cats.Eq

val headsD6D6taildD10D10 = monad.flatMap(fairCoin) { side =>
  if( Eq[Symbol].eqv(side, head) ) {
    monad.flatMap(d6) { a => monad.map(d6) { b => a + b } }
  } else {
    monad.flatMap(d10) { a => monad.map(d10) { b => a + b } }
  }
}
```

Create visualization

```
val iffyChart
 = BarChart[Int, Rational, ConditionalProbabilityTable[Int, Rational], String](
  () => headsD6D6taildD10D10,
  colorOf = _ => Color.blue,
  xAxis = Some(Rational(0)),
  title = Some("if heads, d6+d6, else d10+d10"),
  labelAngle = Some(0d *: angleDouble.degree),
  drawKey = false)
```

Create the SVG

```
iffyChart.svg[IO]("docwork/images/iffy.svg").unsafeRunSync()
```



Further Reading

Motiviating the Monad typeclass is out of scope of this document. Please see the functional programming literature for more about monads and their relationship to functors, applicative functors, monoids, categories, and other structures.

For some historical reading on the origins of probability monads, see the literature on the Giry Monad.

Future work

Measure Theory

Further refining and extending Axle to incorporate Measure Theory is a likely follow-on step.

Markov Categories

As an alternative to Measure Theory, see Tobias Fritz's work on Markov Categories

Probabilistic and Differentiable Programming

In general, the explosion of work on probabilistic and differentible programming is fertile ground for Axle's lawful approach.

Statistics

Common imports and implicits

```
import cats.implicits._
import spire.algebra._
import axle.probability._

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
```

Uniform Distribution

Example

```
val X = uniformDistribution(List(2d, 4d, 4d, 4d, 5d, 5d, 7d, 9d))
// X: ConditionalProbabilityTable[Double, spire.math.Rational] =
 ConditionalProbabilityTable(
//   p = HashMap(5.0 -> 1/4, 9.0 -> 1/8, 2.0 -> 1/8, 7.0 -> 1/8, 4.0 -> 3/8)
// )
```

Standard Deviation

Example

```
import axle.stats._

implicit val nrootDouble: NRoot[Double] = spire.implicits.DoubleAlgebra
```

```
standardDeviation(X)
// res19: Double = 2.0
```

See also **Probability Model**

Root-mean-square deviation

See the Wikipedia page on **Root-mean-square deviation**.

```
import cats.implicits._

import spire.algebra.Field
import spire.algebra.NRoot

import axle.stats._

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
implicit val nrootDouble: NRoot[Double] = spire.implicits.DoubleAlgebra
```

Given four numbers and an estimator function, compute the RMSD:

```
val data = List(1d, 2d, 3d, 4d)
```

```
def estimator(x: Double): Double =
  x + 0.2

rootMeanSquareDeviation[List, Double](data, estimator)
// res21: Double = 0.4000000000000002
```

Reservoir Sampling

Reservoir Sampling is the answer to a common interview question.

```
import spire.random.Generator.rng
import spire.algebra.Field

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra

import axle.stats._
```

Demonstrate it uniformly sampling 15 of the first 100 integers

```
val sample = reservoirSampleK(15, LazyList.from(1), rng).drop(100).head
// sample: List[Int] = List(
//   88,
//   86,
//   81,
//   78,
//   74,
//   65,
//   63,
//   57,
//   54,
//   48,
//   43,
//   41,
//   35,
//   33,
//   2
// )
```

The mean of the sample should be in the ballpark of the mean of the entire list (50.5):

```
import axle.math.arithmeticMean

arithmeticMean(sample.map(_.toDouble))
// res23: Double = 56.53333333333333
```

Indeed it is.

Information Theory

Entropy

The calculation of the entropy of a distribution is available as a function called `entropy` as well as the traditional `H`:

Imports and implicits

```
import edu.uci.ics.jung.graph.DirectedSparseGraph

import cats.implicits._

import spire.math._
import spire.algebra._

import axle._
import axle.probability._
import axle.stats._
import axle.quanta.Information
import axle.jung.directedGraphJung
import axle.data.Coin
import axle.game.Dice.die

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra

implicit val informationConverter
 = Information.converterGraphK2[Double, DirectedSparseGraph]
```

Usage

Entropy of fair 6-sided die

```
val d6 = die(6)
// d6: ConditionalProbabilityTable[Int, Rational] =
 ConditionalProbabilityTable(
//   p = HashMap(5 -> 1/6, 1 -> 1/6, 6 -> 1/6, 2 -> 1/6, 3 -> 1/6, 4 -> 1/6)
// )

H[Int, Rational](d6).show
// res25: String = "2.584962500721156 b"
```

Entropy of fair and biased coins

```
val fairCoin = Coin.flipModel()
// fairCoin: ConditionalProbabilityTable[Symbol, Rational] =
 ConditionalProbabilityTable(
//   p = Map('HEAD -> 1/2, 'TAIL -> 1/2)
// )

H[Symbol, Rational](fairCoin).show
// res26: String = "1.0 b"

val biasedCoin = Coin.flipModel(Rational(7, 10))
// biasedCoin: ConditionalProbabilityTable[Symbol, Rational] =
 ConditionalProbabilityTable(
//   p = Map('HEAD -> 7/10, 'TAIL -> 3/10)
// )

entropy[Symbol, Rational](biasedCoin).show
// res27: String = "0.8812908992306927 b"
```

See also the following example of the entropy of a biased coin.

Example: Entropy of a Biased Coin

Visualize the relationship of a coin's bias to its entropy with this code snippet.

Imports and implicits:

```
import scala.collection.immutable.TreeMap
import cats.implicits._
import spire.math.Rational
import spire.algebra._
import axle.stats.H
import axle.data.Coin
import axle.quanta.UnittedQuantity
import axle.quanta.Information

type D = TreeMap[Rational, UnittedQuantity[Information, Double]]

import edu.uci.ics.jung.graph.DirectedSparseGraph
import axle.jung.directedGraphJung
import cats.kernel.Order
import axle.quanta.unittedTics

implicit val id = {
  implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
  Information.converterGraphK2[Double, DirectedSparseGraph]
}

implicit val or: Order[Rational] = new cats.kernel.Order[Rational] {
  implicit val doubleOrder = Order.fromOrdering[Double]
```

```
  def compare(x: Rational, y: Rational): Int
 = doubleOrder.compare(x.toDouble, y.toDouble)
}
implicit val bitDouble = id.bit
```

Create dataset

```
val hm: D =
  new TreeMap[Rational, UnittedQuantity[Information, Double]]() ++
    (0 to 100).map({ i =>
      val r = Rational(i.toLong, 100L)
      r -> H[Symbol, Rational](Coin.flipModel(r))
    }).toMap
```

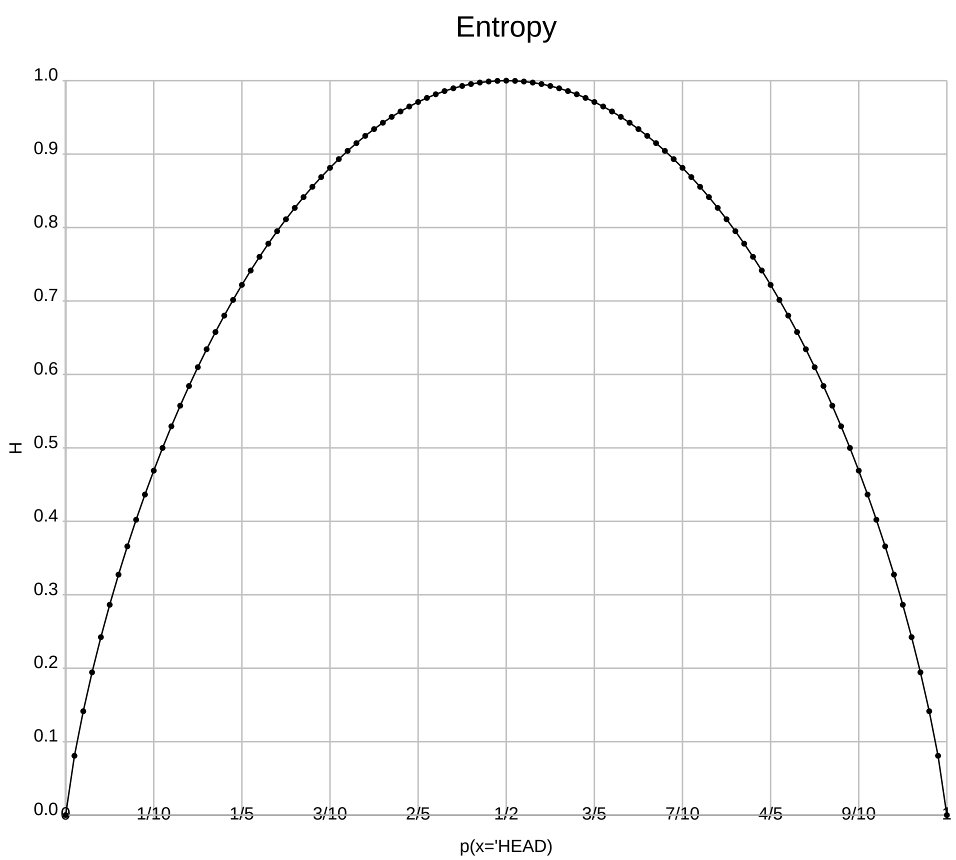
Define visualization

```
import axle.visualize._

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra

val plot = Plot[String, Rational, UnittedQuantity[Information, Double], D](
  () => List(("h", hm)),
  connect = true,
  colorOf = _ => Color.black,
  drawKey = false,
  xAxisLabel = Some("p(x='HEAD)"),
  yAxisLabel = Some("H"),
  title = Some("Entropy")).zeroAxes
```

Create the SVG

```
import axle.web._
import cats.effect._

plot.svg[IO]("docwork/images/coinentropy.svg").unsafeRunSync()
```

The result is the classic Claude Shannon graph



Probabilistic Graphical Models

Currently only Bayesian Networks

Eventually others including Pearl's causal models.

Bayesian Networks

See the Wikipedia page on **Bayesian networks**

Example: Alarm

Define random variables

```
import axle.probability._

val bools = Vector(true, false)

val B = Variable[Boolean]("Burglary")
val E = Variable[Boolean]("Earthquake")
val A = Variable[Boolean]("Alarm")
val J = Variable[Boolean]("John Calls")
val M = Variable[Boolean]("Mary Calls")
```

Define Factor for each variable

```
import spire.math._
import cats.implicits._

val bFactor =
  Factor(Vector(B -> bools), Map(
    Vector(B is true) -> Rational(1, 1000),
    Vector(B is false) -> Rational(999, 1000)))

val eFactor =
  Factor(Vector(E -> bools), Map(
    Vector(E is true) -> Rational(1, 500),
    Vector(E is false) -> Rational(499, 500)))

val aFactor =
  Factor(Vector(B -> bools, E -> bools, A -> bools), Map(
    Vector(B is false, E is false, A is true) -> Rational(1, 1000),
    Vector(B is false, E is false, A is false) -> Rational(999, 1000),
    Vector(B is true, E is false, A is true) -> Rational(940, 1000),
    Vector(B is true, E is false, A is false) -> Rational(60, 1000),
    Vector(B is false, E is true, A is true) -> Rational(290, 1000),
    Vector(B is false, E is true, A is false) -> Rational(710, 1000),
    Vector(B is true, E is true, A is true) -> Rational(950, 1000),
    Vector(B is true, E is true, A is false) -> Rational(50, 1000)))

val jFactor =
  Factor(Vector(A -> bools, J -> bools), Map(
    Vector(A is true, J is true) -> Rational(9, 10),
    Vector(A is true, J is false) -> Rational(1, 10),
    Vector(A is false, J is true) -> Rational(5, 100),
    Vector(A is false, J is false) -> Rational(95, 100)))

val mFactor =
  Factor(Vector(A -> bools, M -> bools), Map(
    Vector(A is true, M is true) -> Rational(7, 10),
    Vector(A is true, M is false) -> Rational(3, 10),
    Vector(A is false, M is true) -> Rational(1, 100),
    Vector(A is false, M is false) -> Rational(99, 100)))
```

Arrange into a graph

```
import axle.pgm._
import axle.jung._
import edu.uci.ics.jung.graph.DirectedSparseGraph

// edges: ba, ea, aj, am

val bn: BayesianNetwork[Boolean, Rational, DirectedSparseGraph[BayesianNetworkNode[Boolean, Rat
 =
  BayesianNetwork.withGraphK2[Boolean, Rational, DirectedSparseGraph](
```

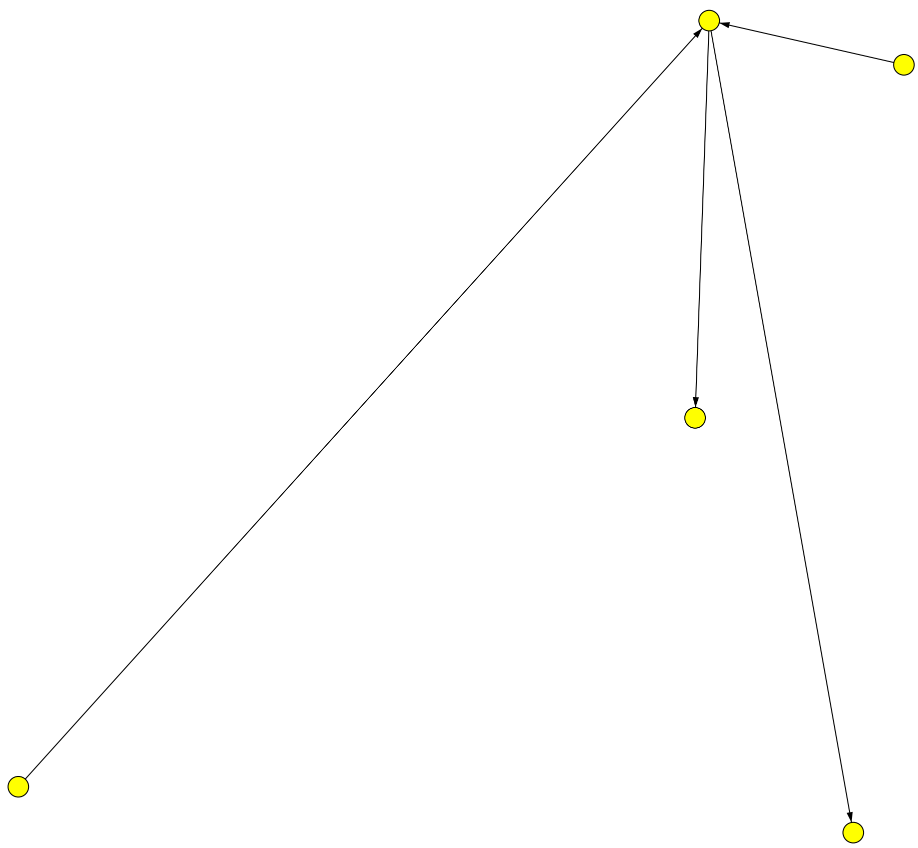
```
    Map(
      B -> bFactor,
      E -> eFactor,
      A -> aFactor,
      J -> jFactor,
      M -> mFactor))
```

Create an SVG visualization

```
import axle.visualize._

val bnVis  = BayesianNetworkVisualization(bn, 1000, 1000, 20)
```

Render as SVG file

```
import axle.web._
import cats.effect._

bnVis.svg[IO]("docwork/images/alarm_bayes.svg").unsafeRunSync()
```



The network can be used to compute the joint probability table:

```
import axle.math.showRational

val jpt = bn.jointProbabilityTable
```

```
jpt.show
// res32: String = """John Calls Alarm Earthquake Burglary Mary Calls
// true       true  true       true     true       1197/1000000000
// true       true  true       true     false      513/1000000000
// true       true  true       false    true       1825173/5000000000
// true       true  true       false    false      782217/5000000000
// true       true  false      true     true       1477539/2500000000
// true       true  false      true     false      633231/2500000000
// true       true  false      false    true       31405563/50000000000
// true       true  false      false    false      13459527/50000000000
// true       false true       true     true       1/20000000000
// true       false true       true     false      99/20000000000
// true       false true       false    true       70929/100000000000
// true       false true       false    false      7021971/100000000000
```

```
// true        false false      true     true        1497/50000000000
// true        false false      true     false       148203/50000000000
// true        false false      false    true        498002499/1000000000000
// true        false false      false    false       49302247401/1000000000000
// false       true  true       true     true        133/1000000000
// false       true  true       true     false       57/1000000000
// false       true  true       false    true        202797/5000000000
// false       true  true       false    false       86913/5000000000
// false       true  false      true     true        164171/2500000000
// false       true  false      true     false       70359/2500000000
// false       true  false      false    true        3489507/50000000000
// false       true  false      false    false       1495503/50000000000
// false       false true       true     true        19/20000000000
// false       false true       true     false       1881/20000000000
// false       false true       false    true        1347651/100000000000
// false       false true       false    false       133417449/100000000000
// false       false false      true     true        28443/50000000000
// false       false false      true     false       2815857/50000000000
// false       false false      false    true        9462047481/1000000000000
// false       false false      false    false
 936742700619/1000000000000"""
```

Variables can be summed out of the factor:

```
import axle._

jpt.sumOut(M).sumOut(J).sumOut(A).sumOut(B).sumOut(E)
// res33: Factor[Boolean, Rational] = Factor(
//   variablesWithValues = Vector(),
//   probabilities = Map(Vector() -> 1)
// )
```

Also written as:

```
jpt.Σ(M).Σ(J).Σ(A).Σ(B).Σ(E)
// res34: Factor[Boolean, Rational] = Factor(
//   variablesWithValues = Vector(),
//   probabilities = Map(Vector() -> 1)
// )
```

Multiplication of factors also works:

```
import spire.implicits.multiplicativeSemigroupOps

val f = (bn.factorFor(A) * bn.factorFor(B)) * bn.factorFor(E)
```

```
f.show
// res35: String = """Burglary Earthquake Alarm
// true     true       true  19/10000000
// true     true       false 1/10000000
// true     false      true  23453/25000000
// true     false      false 1497/25000000
// false    true       true  28971/50000000
// false    true       false 70929/50000000
// false    false      true  498501/500000000
// false    false      false 498002499/500000000"""
```

Markov assumptions:

```
bn.markovAssumptionsFor(M).show
// res36: String = "I({Mary Calls}, {Alarm}, {John Calls, Earthquake,
 Burglary})"
```

This is read as "M is independent of E, B, and J given A".

Probability Model Future Work

Later

- `{CPT,TD}.tailRecM` then ScalaCheck `Monad[CPT,TD]`
- Functor for CPT, TD
- `SigmaAlgebra` for the CPT
- Clean up expressions like `RegionIf[TWOROLLS](_._1 == '#)`
- Laws for `Region` ("Sigma Algebra"? video)
- `OrderedRegion` for the `Order` used in `RegionLTE` and `RegionGTE`?
- Measure Theory
- Test: start with `ABE.jointProbabilityTable` (monotype `tuple5[Boolean]`)
- Factor out each variable until original 5-note network is reached
- Basically the inverse of factor multiplication
- `bn.factorFor(B) * bn.factorFor(E)` should be defined? (It errors)

- `MonotypeBayesanNetwork.filter` collapase into a single BNN
- Rename `ConditionalProbabilityTable`?
- Laws for `Factor`
- Review `InteractionGraph`, `EliminationGraph`, `JoinTree` and the functions they power
- Consider a "case" to be a `Map` vs a `Vector`
- Consider usefulness of `Factor` in terms of `Region`
- `MonotypeBayesanNetwork`.{`pure`, `map`, `flatMap`, `tailRecR`}
- Reconcile `MBN` `combine1` & `combine2`
- Monad tests for `MonotypeBayesanNetwork[Alarm-Burglary-Earthquake]`
- `Bayes[MonotypeBayesanNetwork]` -- could be viewed as "belief updating" (vs "conditioning")

- If it took a ProbabilityModel itself
- Bettings odds
- Multi-armed bandit
- Recursive grid search
- P-values
- z & t scores
- Correlation
- Regression
- Accuracy, Precision
- Bias, Variance
- Cohen's Kappa
- Rm throws from axle.stats.TallyDistribution
- do-calculus (Causality)
- Stochastic Lambda Calculus
- Abadi Plotkin pathology
- Jacobian Vector Products (JVP)
- FLDR probability

- probcomp github

- MIT FSAAD slides

Docs

- Reorder Probability mdoc (Creation, Kolmogorov/Region, Sampler, Bayes, Monad)?

- Footnotes (Giry, etc)

Game Theory

Framework for expressing arbitrary games.

Monty Hall

See the Wikipedia page on the **Monty Hall problem**

The `axle.game.OldMontyHall` object contains a model of the rules of the game.

```
import spire.math.Rational

import axle.probability._
import axle.game.OldMontyHall._
```

The models supports querying the chance of winning given the odds that the player switches his or her initial choice.

At one extreme, the odds of winning given that the other door is always chosen:

```
chanceOfWinning(Rational(1))
// res0: Rational = 2/3
```

At the other extreme, the player always sticks with the initial choice.

```
chanceOfWinning(Rational(0))
// res1: Rational = 1/3
```

The newer `axl.game.montyhall._` package uses `axle.game` typeclasses to model the game:

```
import axle.game._
import axle.game.montyhall._

val game = MontyHall()
```

Create a `writer` for each player that prefixes the player id to all output.

```
import cats.effect.IO
import axle.IO.printMultiLinePrefixed

val playerToWriter: Map[Player, String => IO[Unit]] =
  evGame.players(game).map { player =>
    player -> (printMultiLinePrefixed[IO](player.id) _)
  } toMap
```

Use a uniform distribution on moves as the demo strategy:

```
val randomMove: MontyHallState => ConditionalProbabilityTable[MontyHallMove,
 Rational] =
  (state: MontyHallState) =>
    ConditionalProbabilityTable.uniform[MontyHallMove, Rational]
(evGame.moves(game, state))
```

Wrap the strategies in the calls to `writer` that log the transitions from state to state.

```
val strategies: Player => MontyHallState
 => IO[ConditionalProbabilityTable[MontyHallMove, Rational]] =
  (player: Player) =>
    (state: MontyHallState) =>
      for {
        _ <- playerToWriter(player)
(evGameIO.displayStateTo(game, state, player))
        move <- randomMove.andThen( m => IO { m })(state)
      } yield move
```

Play the game -- compute the end state from the start state.

```
import spire.random.Generator.rng

val endState: MontyHallState =
  play(game, strategies, evGame.startState(game), rng).unsafeRunSync()
// M> Door #1: ???
// M> Door #2: ???
// M> Door #3: ???
// C> Door #1:
// C> Door #2:
// C> Door #3:
// M> Door #1: goat
// M> Door #2: car, first choice
// M> Door #3: goat
// C> Door #1:
// C> Door #2: first choice
// C> Door #3: , revealed goat
// endState: MontyHallState = MontyHallState(
//   placement = Some(value = PlaceCar(door = 2)),
//   placed = true,
//   firstChoice = Some(value = FirstChoice(door = 2)),
//   reveal = Some(value = Reveal(door = 3)),
//   secondChoice = Some(value = Right(value = Stay()))
// )
```

Display outcome to each player

```
val outcome: MontyHallOutcome =
  evGame.mover(game, endState).swap.toOption.get
// outcome: MontyHallOutcome = MontyHallOutcome(car = true)

evGame.players(game).foreach { player =>
  playerToWriter(player)
(evGameIO.displayOutcomeTo(game, outcome, player)).unsafeRunSync()
}
// C> You won the car!
// M> Contestant won the car!
```

Poker

An N-Player, Imperfect Information, Zero-sum game

Poker Analytics Example

The `axle.game.cards` package models decks, cards, ranks, suits, and ordering.

Define a function that takes the hand size and returns the best 5-card hand

```
import cats.implicits._
import cats.Order.catsKernelOrderingForOrder

import axle.game.cards.Deck
import axle.game.poker.PokerHand

def winnerFromHandSize(handSize: Int) =
  Deck().cards.take(handSize).combinations(5).map(cs
 => PokerHand(cs.toVector)).toList.max
```

```
winnerFromHandSize(7).show
// res4: String = "4♠ 4# T# J# A♣"
```

20 simulated 5-card hands made of 7-card hands. Sorted.

```
val hands = (1 to 20).map(n => winnerFromHandSize(7)).sorted
```

```
hands.map({ hand => hand.show + "  " + hand.description }).mkString("\n")
// res5: String = """6# 8# T♣ Q♣ K#  high K high
// 6♣ 8# T♠ Q# K♣  high K high
// 9# T♣ J# Q♠ A#  high A high
// 9♣ T♣ J# K# A♣  high A high
// 4♠ 4♣ J# K♣ A#  pair of 4
// 6# 6♣ T# Q# K#  pair of 6
// 9♣ T♠ T# Q# K#  pair of T
// 9♠ T♠ J# J# A♠  pair of J
```

```
// 7# T♠ J# Q# Q#  pair of Q
// 5♠ 8# K♣ A# A♣  pair of A
// 9# J# K# A# A♣  pair of A
// 2# 2♣ 3♠ 3# 9♣  two pair 3 and 2
// 2# 2# 5♠ 5# J#  two pair 5 and 2
// 8# 8♣ 9# T# T#  two pair T and 8
// 5♠ 5# Q# K# K♣  two pair K and 5
// Q♠ Q♣ K# K♣ A♣  two pair K and Q
// 3♠ 3# J♠ A♠ A#  two pair A and 3
// 9# J♠ J# A# A♣  two pair A and J
// 2# 3# 4# 5♠ 6#  straight to 6
// 3♠ 4# 5# 6# 7♠  straight to 7"""
```

Record 1000 simulated hands for each drawn hand size from 5 to 9

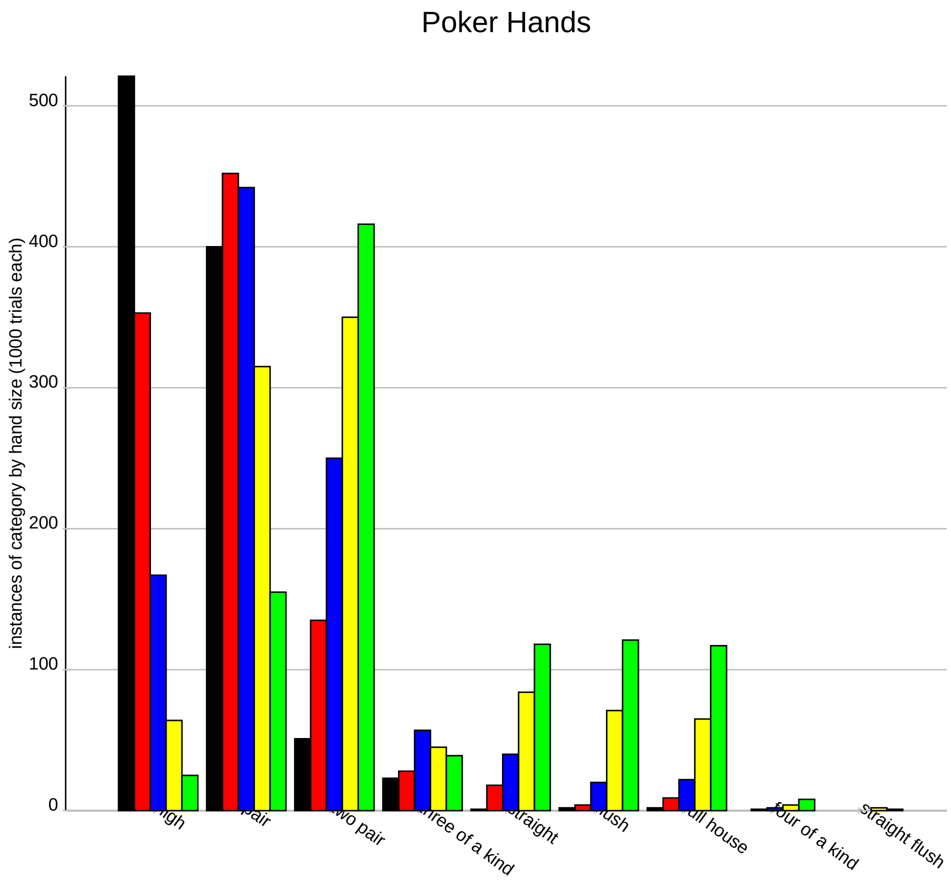
```
import axle.game.poker.PokerHandCategory

val data: IndexedSeq[(PokerHandCategory, Int)] =
  for {
    handSize <- 5 to 9
    trial <- 1 to 1000
  } yield (winnerFromHandSize(handSize).category, handSize)
```

BarChartGrouped to visualize the results

```
import spire.algebra.CRing

import axle.visualize.BarChartGrouped
import axle.visualize.Color._
import axle.syntax.talliable.talliableOps

implicit val ringInt: CRing[Int] = spire.implicits.IntAlgebra

val colors = List(black, red, blue, yellow, green)

val chart
 = BarChartGrouped[PokerHandCategory, Int, Int, Map[(PokerHandCategory, Int), Int], String]
(
  () => data.tally.withDefaultValue(0),
  title = Some("Poker Hands"),
  drawKey = false,
  yAxisLabel = Some("instances of category by hand size (1000 trials each)"),
  colorOf = (cat: PokerHandCategory, handSize: Int) => colors( (handSize - 5)
 % colors.size),
  hoverOf = (cat: PokerHandCategory, handSize: Int) => Some(s"${cat.show}
 from $handSize")
)
```

Render as SVG file

```
import axle.web._
import cats.effect._

chart.svg[IO]("docwork/images/poker_hands.svg").unsafeRunSync()
```



Playing Texas Hold 'Em Poker

As a game of "imperfect information", poker introduces the concept of Information Set.

```
import axle.game._
import axle.game.poker._

val p1 = Player("P1", "Player 1")
val p2 = Player("P2", "Player 2")

val game = Poker(Vector(p1, p2))
```

Create a `writer` for each player that prefixes the player id to all output.

```
import cats.effect.IO
import axle.IO.printMultiLinePrefixed

val playerToWriter: Map[Player, String => IO[Unit]] =
  evGame.players(game).map { player =>
    player -> (printMultiLinePrefixed[IO](player.id) _)
  } toMap
```

Use a uniform distribution on moves as the demo strategy:

```
import axle.probability._
import spire.math.Rational

val randomMove =
  (state: PokerStateMasked) =>
    ConditionalProbabilityTable.uniform[PokerMove, Rational]
(evGame.moves(game, state))
```

Wrap the strategies in the calls to `writer` that log the transitions from state to state.

```
val strategies: Player => PokerStateMasked
 => IO[ConditionalProbabilityTable[PokerMove, Rational]] =
  (player: Player) =>
    (state: PokerStateMasked) =>
      for {
        _ <- playerToWriter(player)
(evGameIO.displayStateTo(game, state, player))
        move <- randomMove.andThen( m => IO { m })(state)
      } yield move
```

Play the game -- compute the end state from the start state.

```
import spire.random.Generator.rng

val endState
 = play(game, strategies, evGame.startState(game), rng).unsafeRunSync()
// D> To: You
// D> Current bet: 0
// D> Pot: 0
// D> Shared:
// D>
// D> P1:  hand -- in for $--, $100 remaining
// D> P2:  hand -- in for $--, $100 remaining
// P1> To: You
// P1> Current bet: 2
// P1> Pot: 3
// P1> Shared:
// P1>
// P1> P1:  hand T♣ 7# in for $1, $99 remaining
```

```
// P1> P2:  hand -- in for $2, $98 remaining
// P2> To: You
// P2> Current bet: 96
// P2> Pot: 98
// P2> Shared:
// P2>
// P2> P1:  hand -- in for $96, $4 remaining
// P2> P2:  hand J# 8# in for $2, $98 remaining
// P1> To: You
// P1> Current bet: 100
// P1> Pot: 196
// P1> Shared:
// P1>
// P1> P1:  hand T♣ 7# in for $96, $4 remaining
// P1> P2:  hand -- in for $100, $0 remaining
// D> To: You
// D> Current bet: 100
// D> Pot: 196
// D> Shared:
// D>
// D> P1:  hand -- out, $4 remaining
// D> P2:  hand -- in for $100, $0 remaining
// endState: PokerState = PokerState(
//   moverFn = axle.game.poker.package$$anon$1$$Lambda
$10153/0x00000008026b1c48@4662cba2,
//   deck = Deck(
//     cards = List(
//       Card(
//         rank = axle.game.cards.Ace$@609f6ed7,
//         suit = axle.game.cards.Diamonds$@2838fa45
//       ),
//       Card(
//         rank = axle.game.cards.R6$@2f1286c4,
//         suit = axle.game.cards.Clubs$@1d288b46
//       ),
//       Card(
//         rank = axle.game.cards.Ace$@609f6ed7,
//         suit = axle.game.cards.Spades$@5dc42049
//       ),
//       Card(
//         rank = axle.game.cards.R3$@69600d6e,
//         suit = axle.game.cards.Hearts$@5b00e7d5
//       ),
//       Card(
//         rank = axle.game.cards.R3$@69600d6e,
//         suit = axle.game.cards.Spades$@5dc42049
//       ),
//       Card(
//         rank = axle.game.cards.R4$@4017bbb1,
//         suit = axle.game.cards.Diamonds$@2838fa45
//       ),
//       Card(
//         rank = axle.game.cards.R4$@4017bbb1,
```

```
//          suit = axle.game.cards.Spades$@5dc42049
//        ),
//        Card(
//          rank = axle.game.cards.R4$@4017bbb1,
//          suit = axle.game.cards.Clubs$@1d288b46
//        ),
//        Card(
//          rank = axle.game.cards.R7$@7d441f09,
//          suit = axle.game.cards.Diamonds$@2838fa45
//        ),
//        Card(
//          rank = axle.game.cards.R5$@6da34b54,
//          suit = axle.game.cards.Clubs$@1d288b46
//        ),
//        Card(
//          rank = axle.game.cards.King$@48cb9f74,
//          suit = axle.game.cards.Hearts$@5b00e7d5
//        ),
// ...
```

Display outcome to each player

```
val outcome = evGame.mover(game, endState).swap.toOption.get
// outcome: PokerOutcome = PokerOutcome(
//   winner = Some(value = Player(id = "P2", description = "Player 2")),
//   hand = None
// )

evGame.players(game).foreach { player =>
  playerToWriter(player)
(evGameIO.displayOutcomeTo(game, outcome, player)).unsafeRunSync()
}
// D> Winner: Player 2
// D> Hand   : not shown
// P1> Winner: Player 2
// P1> Hand   : not shown
// P2> Winner: Player 2
// P2> Hand   : not shown
```

Prisoner's Dilemma

See the Wikipedia page on the **Prisoner's Dilemma**

The `axl.game.prisoner._` package uses `axle.game` typeclasses to model the game:

```
import axle.game._
import axle.game.prisoner._

val p1 = Player("P1", "Player 1")
val p2 = Player("P2", "Player 2")

val game = PrisonersDilemma(p1, p2)
```

Create a `writer` for each player that prefixes the player id to all output.

```
import cats.effect.IO
import axle.IO.printMultiLinePrefixed

val playerToWriter: Map[Player, String => IO[Unit]] =
  evGame.players(game).map { player =>
    player -> (printMultiLinePrefixed[IO](player.id) _)
  } toMap
```

Use a uniform distribution on moves as the demo strategy:

```
import axle.probability._
import spire.math.Rational

val randomMove =
  (state: PrisonersDilemmaState) =>
    ConditionalProbabilityTable.uniform[PrisonersDilemmaMove, Rational]
(evGame.moves(game, state))
```

Wrap the strategies in the calls to `writer` that log the transitions from state to state.

```
val strategies: Player => PrisonersDilemmaState
 => IO[ConditionalProbabilityTable[PrisonersDilemmaMove, Rational]] =
  (player: Player) =>
    (state: PrisonersDilemmaState) =>
      for {
        _ <- playerToWriter(player)
(evGameIO.displayStateTo(game, state, player))
        move <- randomMove.andThen( m => IO { m })(state)
      } yield move
```

Play the game -- compute the end state from the start state.

```
import spire.random.Generator.rng

val endState
 = play(game, strategies, evGame.startState(game), rng).unsafeRunSync()
// P1> You have been caught
// P2> You have been caught
// endState: PrisonersDilemmaState = PrisonersDilemmaState(
//   p1Move = Some(value = Betrayal()),
//   p1Moved = true,
//   p2Move = Some(value = Betrayal())
// )
```

Display outcome to each player

```
val outcome = evGame.mover(game, endState).swap.toOption.get
// outcome: PrisonersDilemmaOutcome = PrisonersDilemmaOutcome(
//   p1YearsInPrison = 2,
//   p2YearsInPrison = 2
// )

evGame.players(game).foreach { player =>
  playerToWriter(player)
(evGameIO.displayOutcomeTo(game, outcome, player)).unsafeRunSync()
}
// P1> You is imprisoned for 2 years
// P1> Player 2 is imprisoned for 2 years
// P2> Player 1 is imprisoned for 2 years
// P2> You is imprisoned for 2 years
```

Tic-Tac-Toe

A Perfect Information, Zero-sum game

Playing Tic-Tac-Toe

```
import axle.game._
import axle.game.ttt._

val x = Player("X", "Player X")
val o = Player("O", "Player O")

val game = TicTacToe(3, x, o)
```

Create a `writer` for each player that prefixes the player id to all output.

```scala
import cats.effect.IO
import axle.IO.printMultiLinePrefixed

val playerToWriter: Map[Player, String => IO[Unit]] =
  evGame.players(game).map { player =>
    player -> (printMultiLinePrefixed[IO](player.id) _)
  } toMap
```

Use a uniform distribution on moves as the demo strategy:

```scala
import axle.probability._
import spire.math.Rational

val randomMove =
  (state: TicTacToeState) =>
    ConditionalProbabilityTable.uniform[TicTacToeMove, Rational]
(evGame.moves(game, state))
```

Wrap the strategies in the calls to `writer` that log the transitions from state to state.

```scala
val strategies: Player => TicTacToeState
 => IO[ConditionalProbabilityTable[TicTacToeMove, Rational]] =
  (player: Player) =>
    (state: TicTacToeState) =>
      for {
        _ <- playerToWriter(player)
(evGameIO.displayStateTo(game, state, player))
        move <- randomMove.andThen( m => IO { m })(state)
      } yield move
```

Play the game -- compute the end state from the start state.

```scala
import spire.random.Generator.rng

val endState
 = play(game, strategies, evGame.startState(game), rng).unsafeRunSync()
// X> Board:         Movement Key:
// X>  | |           1|2|3
// X>  | |           4|5|6
// X>  | |           7|8|9
// O> Board:         Movement Key:
// O>  |X|           1|2|3
// O>  | |           4|5|6
// O>  | |           7|8|9
// X> Board:         Movement Key:
// X>  |X|O          1|2|3
// X>  | |           4|5|6
// X>  | |           7|8|9
// O> Board:         Movement Key:
```

```
// O>  |X|O          1|2|3
// O>  | |X          4|5|6
// O>  | |           7|8|9
// X> Board:         Movement Key:
// X>  |X|O          1|2|3
// X>  | |X          4|5|6
// X>  |O|           7|8|9
// O> Board:         Movement Key:
// O>  |X|O          1|2|3
// O>  | |X          4|5|6
// O>  |O|X          7|8|9
// X> Board:         Movement Key:
// X>  |X|O          1|2|3
// X> O| |X          4|5|6
// X>  |O|X          7|8|9
// O> Board:         Movement Key:
// O>  |X|O          1|2|3
// O> O|X|X          4|5|6
// O>  |O|X          7|8|9
// X> Board:         Movement Key:
// X>  |X|O          1|2|3
// X> O|X|X          4|5|6
// X> O|O|X          7|8|9
// endState: TicTacToeState = TicTacToeState(
//   moverOpt = None,
//   board = Array(
//     Some(value = Player(id = "X", description = "Player X")),
//     Some(value = Player(id = "X", description = "Player X")),
//     Some(value = Player(id = "O", description = "Player O")),
//     Some(value = Player(id = "O", description = "Player O")),
//     Some(value = Player(id = "X", description = "Player X")),
//     Some(value = Player(id = "X", description = "Player X")),
//     Some(value = Player(id = "O", description = "Player O")),
//     Some(value = Player(id = "O", description = "Player O")),
//     Some(value = Player(id = "X", description = "Player X"))
//   ),
//   boardSize = 3
// )
```

Display outcome to each player

```scala
val outcome = evGame.mover(game, endState).swap.toOption.get
// outcome: TicTacToeOutcome = TicTacToeOutcome(
//   winner = Some(value = Player(id = "X", description = "Player X"))
// )

evGame.players(game).foreach { player =>
  playerToWriter(player)
(evGameIO.displayOutcomeTo(game, outcome, player)).unsafeRunSync()
}
// X> You beat Player O!
// O> Player X beat You!
```

Future Work

Missing functionality

- Remove moveStateStream
- For one game (probably Poker)
- Record witnessed and unwitnessed history `Seq[(M, S)]` in `State`
- Display to user in interactiveMove
    - `val mm = evGame.maskMove(game, move, mover, observer)`
    - `evGameIO.displayMoveTo(game, mm, mover, observer)`
- Then generalize and pull into framework

Motivating Examples

- Generalize `OldMontyHall.chanceOfWinning`
- GuessRiffle.md
- Walk through game
- Plot distribution of sum(entropy) for both strategies
- Plot entropy by turn # for each strategy
- Plot simulated score distribution for each strategy
- GuessRiffleSpec: use `moveFromRandomState`
- Gerrymandering sensitivity

- "You split, I choose" as game

Deeper changes to axle.game

- `aiMover.unmask` prevents `MontyHallSpec` "AI vs. AI game produces moveStateStream" from working

- will be an issue for all non-perfect information
- Identify all uses of `spire.random.Generator` (and other random value generation)
- See uses of `seed` in `GuessRiffleProperties`
- Eliminate entropy consumption of `rng` side-effect (eg `applyMove(Riffle())`)

- `Chance` should be its own player

- Consider whether `PM` should be a part of `Strategy` type (`MS => PM[M, V]`)
  - More abstractly, more many intents and purposes, all we are about is that resolving PM to M consumes entropy
  - In which cases should the `PM` be retained?

- Each N bits consumed during `Riffle()` is its own move

- Chance moves consume `UnittedQuantity[Information, N]`
- `perceive` could return a lower-entropy probability model

- Perhaps in exchange for a given amount of energy

- Or ask for a 0-entropy model and be told how expensive that was
- Game theory axioms (Nash, etc)
- `axle.game`: `Observable[T]`

Hygeine

- performance benchmark
- Replace `axle.game.moveFromRandomState.mapToProb`
- Clean up `axle.game.playWithIntroAndOutcomes`
- The references to `movesMap` in `MoveFromRandomStateSpec.scala` illustrate a need for a cleaner way to create a hard-coded strategy -- which could just be in the form of a couple utility functions from `movesMap` to the data needed by `evGame.{moves,applyMove}` and `rm` strategy

- Generalize `ConditionalProbabilityTable.uniform` into typeclass
- Simplify `GuessRiffleProperties` (especially second property)
- stateStreamMap only used in GuessRiffleProperties -- stop using chain?
- stateStrategyMoveStream only used in GuessRiffleProperties
- `Game.players` should be a part of GameState (or take it as an argument)? Will wait for pressing use case.

Game Theory and Examples

- Game Theory: information sets, equilibria
- Factor `axle.game.moveFromRandomState` in terms of a random walk on a graph.
- See "TODO scale mass down"
- Compare to Brownian motion, Random walk, Ito process, ...
- Provide some axioms
  - no outgoing with path in from non-zero mass monotonically increases
  - no incoming with path out monotonically decreases
- possibly provide a version for acyclic graphs
- Iterative game playing algorithm is intractible, but shares intent with sequential monte carlo
- Think about Information Theory's "Omega" vis-a-vis Sequential Monte Carlo
- Improve `axle.stats.rationalProbabilityDist` as probabilities become smaller
- SimpsonsParadox.md
- Axioms of partial differentiation
- **Plotkin Partial Differentiation**
- Conal Elliott: Efficient automatic differentiation made easy via category theory
- Max bet for Poker
- syntax for `Game` typeclass

Chaos Theory

Mandelbrot Set

See the wikipedia page on the **Mandelbrot Set**

First a couple imports:

```
import cats.implicits._
import spire.algebra.Field
import axle._
import axle.math._
```

Define a function to compute the Mandelbrot velocity at point on the plane `(x, y)`

```
implicit val fieldDouble: Field[Double] =
  spire.implicits.DoubleAlgebra

val f: (Double, Double, Double, Double) => Int =
  (x0: Double, x1: Double, y0: Double, y1: Double) =>
    inMandelbrotSetAt(4d, x0, y0, 1000).getOrElse(-1)
```

Import visualization package

```
import axle.visualize._
```

Define a "velocity to color" function

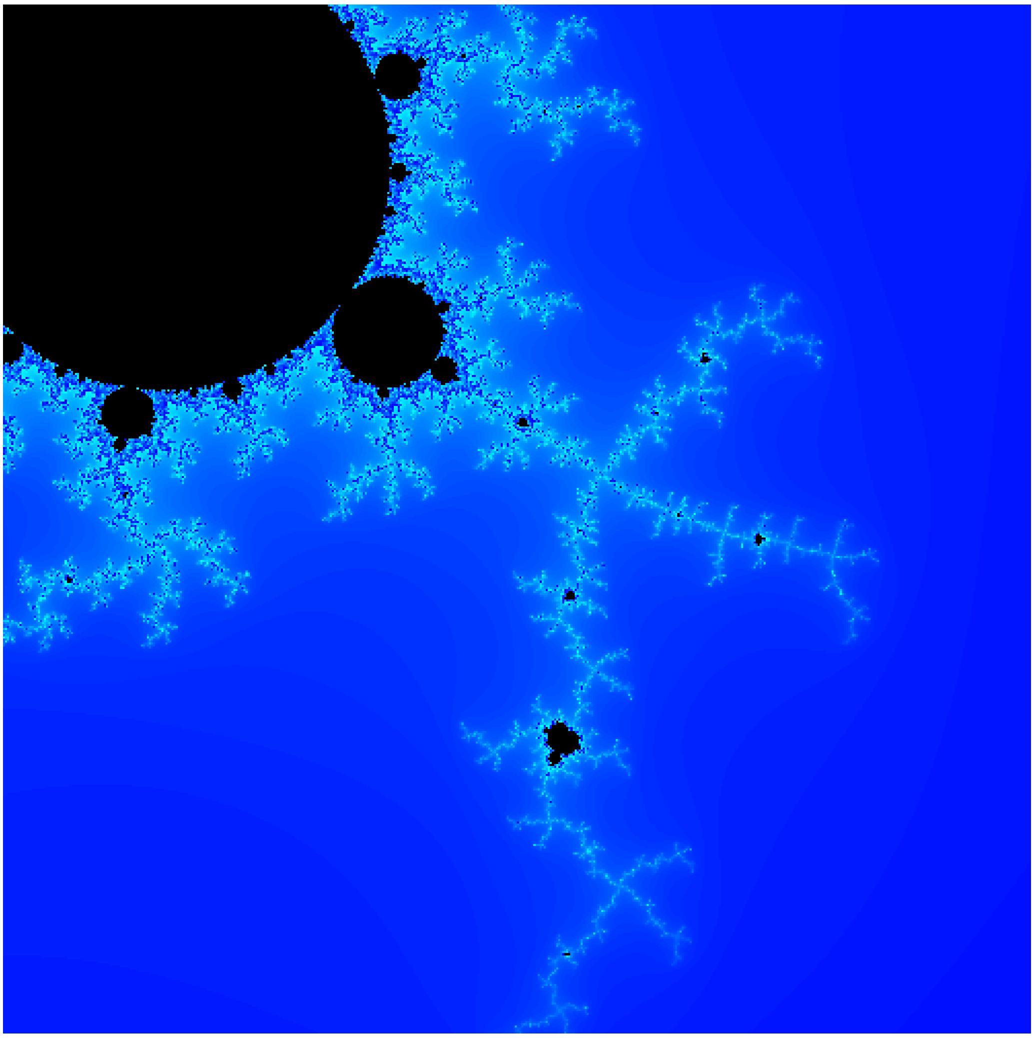
```
val colors = (0 to 255).map(g => Color(0, g, 255)).toArray

val v2c: Int => Color =
  (v: Int) => if( v == -1 ) Color.black else colors((v*5) % 256)
```

Define a `PixelatedColoredArea` to show a range of the Mandelbrot Set.

```
val pca = PixelatedColoredArea[Double, Double, Int](
  f,
  v2c,
  width = 500,
  height = 500,
  minX = 0.25,
  maxX = 0.45,
  minY = 0.50,
  maxY = 0.70)
```

Create PNG

```
import axle.awt._
import cats.effect._

pca.png[IO]("docwork/images/mandelbrot.png").unsafeRunSync()
```



Some other parts of the set to explore:

```
val pca = PixelatedColoredArea(f, v2c, 1600, 1600, 0d, 1d, 0d, 1d)

val pca = PixelatedColoredArea(f, v2c, 1600, 1600, 0d, 0.5, 0.5, 1d)

val pca = PixelatedColoredArea(f, v2c, 1600, 1600, 0.25d, 0.5, 0.5, 0.75d)

val pca = PixelatedColoredArea(f, v2c, 3000, 3000, 0.20d, 0.45, 0.45, 0.70d)
```

Logistic Map

See the wikipedia page on **Logistic Map** function

Create data for a range of the logistic map function

```
import spire.algebra._

val initial = 0.3

import java.util.TreeSet
val memo = collection.mutable.Map.empty[Double, TreeSet[Double]]
implicit val ringDouble: Ring[Double] = spire.implicits.DoubleAlgebra

def lhsContainsMark(minX: Double, maxX: Double, maxY: Double, minY: Double): Boolean
 = {
  val λ = minX
  val f = axle.math.logisticMap(λ)
  val set = memo.get(λ).getOrElse {
    val set = new TreeSet[Double]()
    axle.algebra.applyForever(f, initial).drop(10000).take(200) foreach
 { set.add }
    memo += minX -> set
    set
  }
  !set.tailSet(minY).headSet(maxY).isEmpty
}
```

Define a "value to color" function.

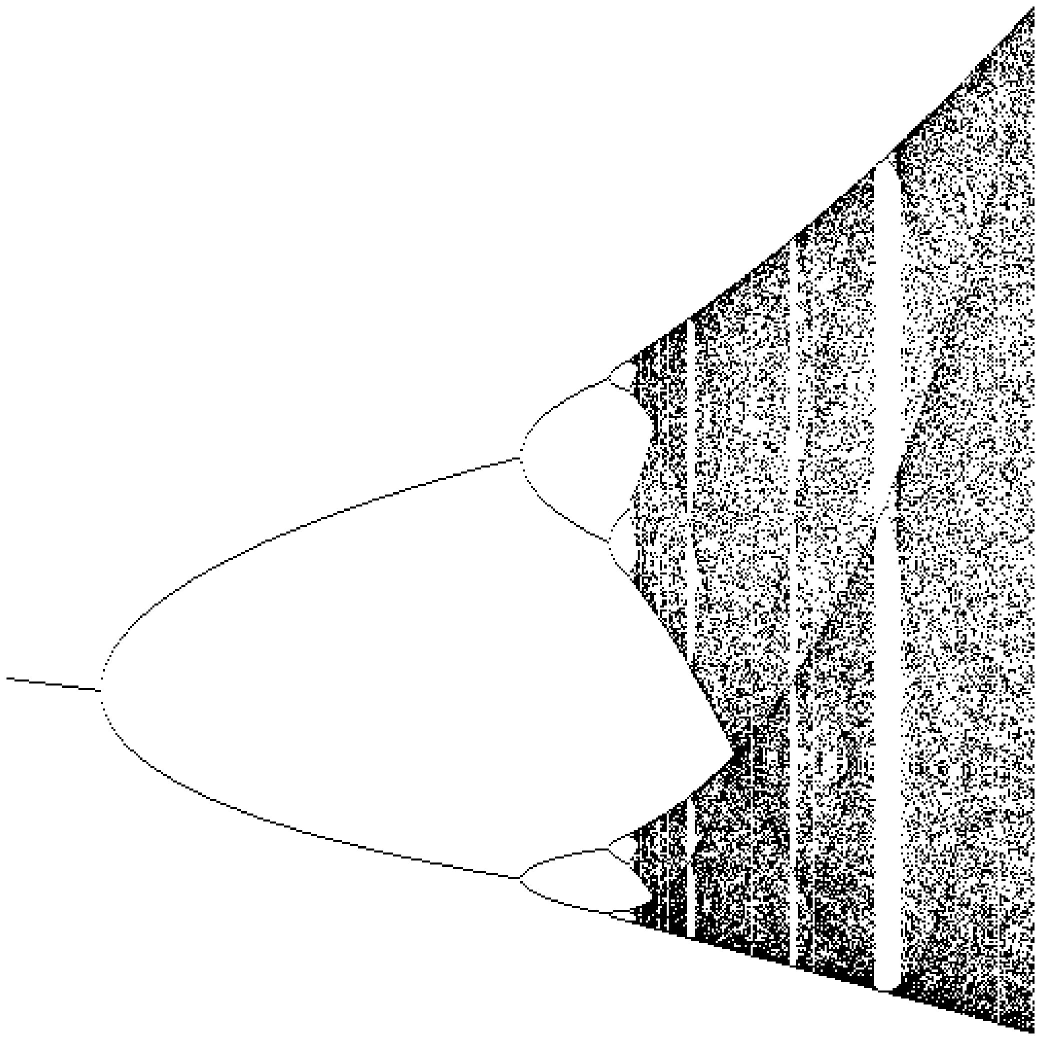
```
import axle.visualize._

val v2c: Boolean => Color =
  (v: Boolean) => if (v) Color.black else Color.white
```

Define a `PixelatedColoredArea` to show a range of Logistic Map.

```
import cats.implicits._

val pca = PixelatedColoredArea[Double, Double, Boolean](
  lhsContainsMark,
  v2c,
  width = 500,
  height = 500,
  minX = 2.9,
  maxX = 4d,
  minY = 0d,
  maxY = 1d
)
```

Create the PNG

```
import axle.awt._
import cats.effect._

pca.png[IO]("docwork/images/logMap.png").unsafeRunSync()
```



Machine Learning

Linear Regression

`axle.ml.LinearRegression` makes use of `axle.algebra.LinearAlgebra`.

See the wikipedia page on **Linear Regression**

Example: Home Prices

```
case class RealtyListing(size: Double, bedrooms: Int, floors: Int, age: Int, price: Double)

val listings = List(
  RealtyListing(2104, 5, 1, 45, 460d),
  RealtyListing(1416, 3, 2, 40, 232d),
  RealtyListing(1534, 3, 2, 30, 315d),
  RealtyListing(852, 2, 1, 36, 178d))
```

Create a price estimator using linear regression.

```
import cats.implicits._
import spire.algebra.Rng
import spire.algebra.NRoot
import axle.jblas._

implicit val rngDouble: Rng[Double] = spire.implicits.DoubleAlgebra
implicit val nrootDouble: NRoot[Double] = spire.implicits.DoubleAlgebra
implicit val laJblasDouble = axle.jblas.linearAlgebraDoubleMatrix[Double]
implicit val rngInt: Rng[Int] = spire.implicits.IntAlgebra

import axle.ml.LinearRegression

val priceEstimator = LinearRegression(
  listings,
  numFeatures = 4,
  featureExtractor = (rl: RealtyListing) => (rl.size :: rl.bedrooms.toDouble
 :: rl.floors.toDouble :: rl.age.toDouble :: Nil),
  objectiveExtractor = (rl: RealtyListing) => rl.price,
  α = 0.1,
  iterations = 100)
```

Use the estimator

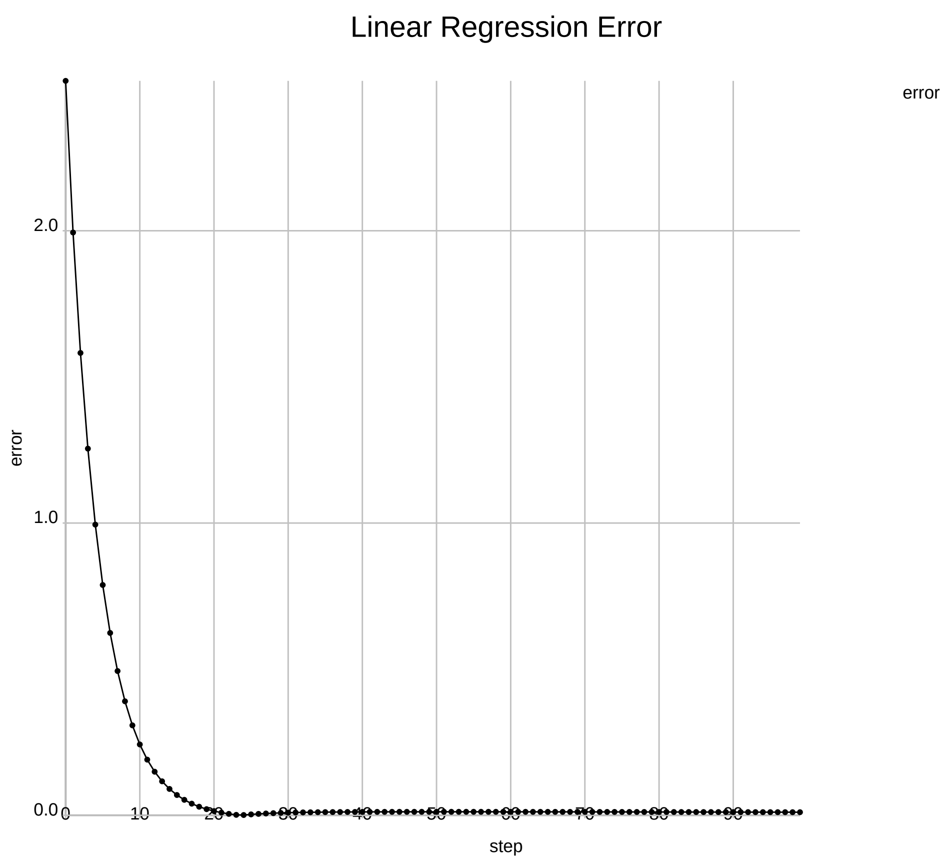
```
priceEstimator(RealtyListing(1416, 3, 2, 40, 0d))
// res0: Double = 288.60017635814035
```

Create a Plot of the error during the training

```
import axle.visualize._
import axle.algebra.Plottable._

val errorPlot = Plot(
  () => List(("error" -> priceEstimator.errTree)),
  connect = true,
  drawKey = true,
  colorOf = (label: String) => Color.black,
  title = Some("Linear Regression Error"),
  xAxis = Some(0d),
  xAxisLabel = Some("step"),
  yAxis = Some(0),
  yAxisLabel = Some("error"))
```

Create the SVG

```
import axle.web._
import cats.effect._

errorPlot.svg[IO]("docwork/images/lrerror.svg").unsafeRunSync()
```



Naive Bayes

Naïve Bayes

Example: Tennis and Weather

```
case class Tennis(outlook: String, temperature: String, humidity: String, wind: String, play: B

val events = List(
  Tennis("Sunny", "Hot", "High", "Weak", false),
  Tennis("Sunny", "Hot", "High", "Strong", false),
  Tennis("Overcast", "Hot", "High", "Weak", true),
  Tennis("Rain", "Mild", "High", "Weak", true),
  Tennis("Rain", "Cool", "Normal", "Weak", true),
  Tennis("Rain", "Cool", "Normal", "Strong", false),
  Tennis("Overcast", "Cool", "Normal", "Strong", true),
  Tennis("Sunny", "Mild", "High", "Weak", false),
  Tennis("Sunny", "Cool", "Normal", "Weak", true),
  Tennis("Rain", "Mild", "Normal", "Weak", true),
  Tennis("Sunny", "Mild", "Normal", "Strong", true),
  Tennis("Overcast", "Mild", "High", "Strong", true),
  Tennis("Overcast", "Hot", "Normal", "Weak", true),
  Tennis("Rain", "Mild", "High", "Strong", false))
```

Build a classifier to predict the Boolean feature 'play' given all the other features of the observations

```
import cats.implicits._

import spire.math._

import axle._
import axle.probability._
import axle.ml.NaiveBayesClassifier
```

```
val classifier = NaiveBayesClassifier[Tennis, String, Boolean, List, Rational](
  events,
  List(
    (Variable[String]("Outlook") -> Vector("Sunny", "Overcast", "Rain")),
    (Variable[String]("Temperature") -> Vector("Hot", "Mild", "Cool")),
    (Variable[String]("Humidity") -> Vector("High", "Normal", "Low")),
    (Variable[String]("Wind") -> Vector("Weak", "Strong"))),
  (Variable[Boolean]("Play") -> Vector(true, false)),
  (t: Tennis) => t.outlook :: t.temperature :: t.humidity :: t.wind :: Nil,
  (t: Tennis) => t.play)
```

Use the classifier to predict:

```
events map { datum => datum.toString + "\t"
 + classifier(datum) } mkString("\n")
// res3: String = """Tennis(Sunny,Hot,High,Weak,false) false
// Tennis(Sunny,Hot,High,Strong,false) false
// Tennis(Overcast,Hot,High,Weak,true) true
// Tennis(Rain,Mild,High,Weak,true) true
// Tennis(Rain,Cool,Normal,Weak,true) true
// Tennis(Rain,Cool,Normal,Strong,false) true
// Tennis(Overcast,Cool,Normal,Strong,true) true
// Tennis(Sunny,Mild,High,Weak,false) false
// Tennis(Sunny,Cool,Normal,Weak,true) true
// Tennis(Rain,Mild,Normal,Weak,true) true
// Tennis(Sunny,Mild,Normal,Strong,true) true
// Tennis(Overcast,Mild,High,Strong,true) true
// Tennis(Overcast,Hot,Normal,Weak,true) true
// Tennis(Rain,Mild,High,Strong,false) false"""
```

Measure the classifier's performance

```
import axle.ml.ClassifierPerformance

ClassifierPerformance[Rational, Tennis, List](events, classifier, _.play).show
// res4: String = """Precision   9/10
// Recall      1
// Specificity 4/5
// Accuracy    13/14
// F1 Score    18/19
// """
```

See **Precision and Recall** for the definition of the performance metrics.

k-Means Clustering

Example: Irises

See the wikipedia page on **k-Means Clustering**

A demonstration of k-Means Clustering using the **Iris flower data set**

Imports for Distance quanta

```
import edu.uci.ics.jung.graph.DirectedSparseGraph
import cats.implicits._
import spire.algebra._
import axle._
import axle.quanta.Distance
import axle.quanta.DistanceConverter
import axle.jung._
```

```
implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra

implicit val distanceConverter = {
  import axle.algebra.modules.doubleRationalModule
  Distance.converterGraphK2[Double, DirectedSparseGraph]
}
```

Import the Irises data set

```
import axle.data.Irises
import axle.data.Iris
```

```
val ec = scala.concurrent.ExecutionContext.global
val blocker = cats.effect.Blocker.liftExecutionContext(ec)
implicit val cs = cats.effect.IO.contextShift(ec)

val irisesIO = new Irises[cats.effect.IO](blocker)
val irises = irisesIO.irises.unsafeRunSync()
```

Make a 2-D Euclidean space implicitly available for clustering

```
import org.jblas.DoubleMatrix
import axle.algebra.distance.Euclidean
import axle.jblas.linearAlgebraDoubleMatrix
import axle.jblas.rowVectorInnerProductSpace

implicit val nrootDouble: NRoot[Double] = spire.implicits.DoubleAlgebra

implicit val space: Euclidean[DoubleMatrix, Double] = {
  implicit val ringInt: Ring[Int] = spire.implicits.IntAlgebra
  implicit val inner = rowVectorInnerProductSpace[Int, Int, Double](2)
  new Euclidean[DoubleMatrix, Double]
}
```

Build a classifier of irises based on sepal length and width using the K-Means algorithm

```
import spire.random.Generator.rng
import axle.ml.KMeans
import axle.ml.PCAFeatureNormalizer
import distanceConverter.cm
```

```
val irisFeaturizer =
  (iris: Iris) => List((iris.sepalLength in cm).magnitude.toDouble,
 (iris.sepalWidth in cm).magnitude.toDouble)

implicit val la = linearAlgebraDoubleMatrix[Double]
```

```
val normalizer = (PCAFeatureNormalizer[DoubleMatrix] _).curried.apply(0.98)

val classifier: KMeans[Iris, List, DoubleMatrix] =
  KMeans[Iris, List, DoubleMatrix](
    irises,
    N = 2,
    irisFeaturizer,
    normalizer,
    K = 3,
    iterations = 20)(rng)
```

Produce a "confusion matrix"

```
import axle.ml.ConfusionMatrix

val confusion = ConfusionMatrix[Iris, Int, String, Vector, DoubleMatrix](
  classifier,
  irises.toVector,
  _.species,
  0 to 2)
```

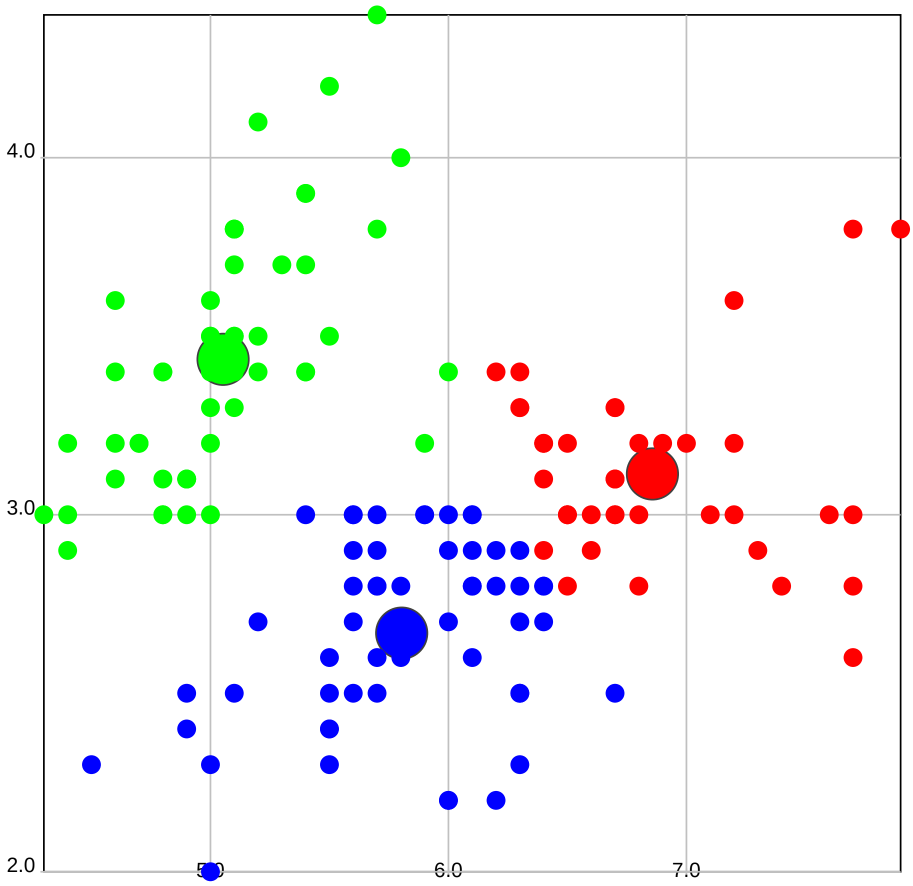
```
confusion.show
// res6: String = """  0   1  49 :  50 Iris-setosa
//  12  36   2 :  50 Iris-versicolor
//  30  20   0 :  50 Iris-virginica
//
//  42  57  51
// """
```

Visualize the final (two dimensional) centroid positions

```
import axle.visualize.KMeansVisualization
import axle.visualize.Color._

val colors = Vector(red, blue, green)

val vis = KMeansVisualization[Iris, List, DoubleMatrix](classifier, colors)
```

Create the SVG

```
import axle.web._
import cats.effect._

vis.svg[IO]("docwork/images/k_means.svg").unsafeRunSync()
```

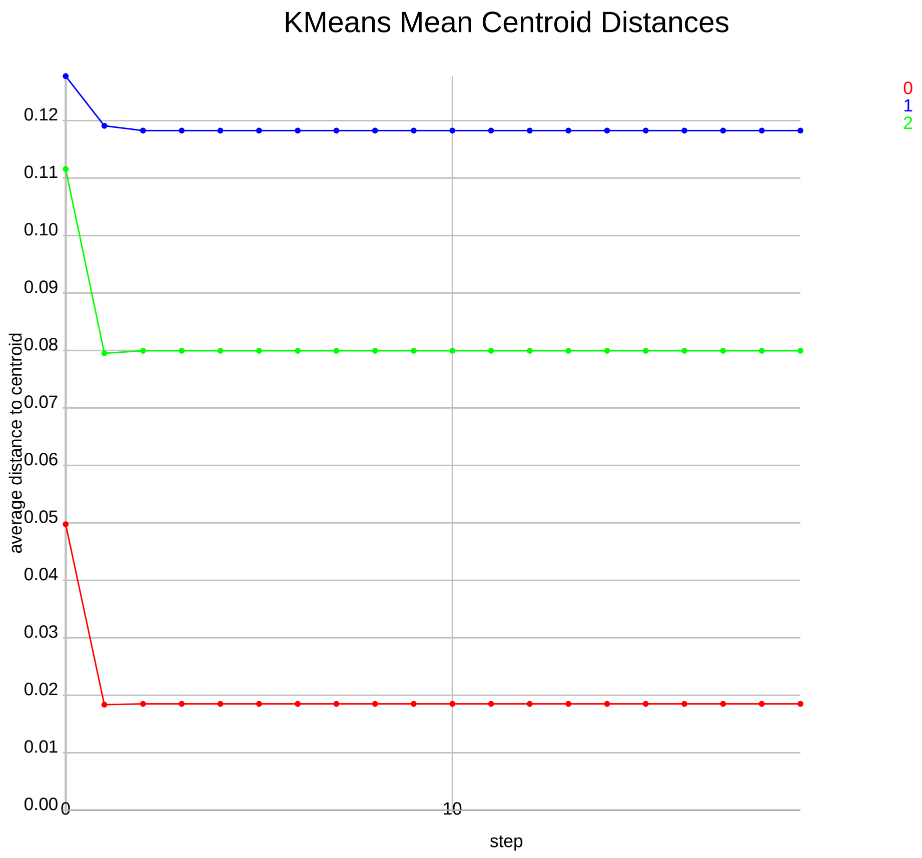


Average centroid/cluster vs iteration:

```
import scala.collection.immutable.TreeMap
import axle.visualize._

val plot = Plot(
  () => classifier.distanceLogSeries,
  connect = true,
  drawKey = true,
  colorOf = colors,
  title = Some("KMeans Mean Centroid Distances"),
  xAxis = Some(0d),
  xAxisLabel = Some("step"),
  yAxis = Some(0),
  yAxisLabel = Some("average distance to centroid"))
```

Create the SVG

```scala
import axle.web._
import cats.effect._

plot.svg[IO]("docwork/images/kmeansvsiteration.svg").unsafeRunSync()
```



Example: Federalist Papers

Imports

```scala
import axle.data.FederalistPapers
import FederalistPapers.Article
```

Download (and cache) the Federalist articles downloader:

```scala
val ec = scala.concurrent.ExecutionContext.global
val blocker = cats.effect.Blocker.liftExecutionContext(ec)
implicit val cs = cats.effect.IO.contextShift(ec)

val articlesIO = FederalistPapers.articles[cats.effect.IO](blocker)

val articles = articlesIO.unsafeRunSync()
```

The result is a `List[Article]`. How many articles are there?

```
articles.size
// res10: Int = 86
```

Construct a `Corpus` object to assist with content analysis

```
import axle.nlp._
import axle.nlp.language.English

import spire.algebra.CRing
implicit val ringLong: CRing[Long] = spire.implicits.LongAlgebra

val corpus = Corpus[Vector, Long](articles.map(_.text).toVector, English)
```

Define a feature extractor using top words and bigrams.

```
val frequentWords = corpus.wordsMoreFrequentThan(100)
// frequentWords: List[String] = List(
//   "the",
//   "of",
//   "to",
//   "and",
//   "in",
//   "a",
//   "be",
//   "that",
//   "it",
//   "is",
//   "which",
//   "by",
//   "as",
// ...
```

```
val topBigrams = corpus.topKBigrams(200)
// topBigrams: List[(String, String)] = List(
//   ("of", "the"),
//   ("to", "the"),
//   ("in", "the"),
//   ("to", "be"),
//   ("that", "the"),
//   ("it", "is"),
//   ("by", "the"),
//   ("of", "a"),
//   ("the", "people"),
//   ("on", "the"),
//   ("would", "be"),
//   ("will", "be"),
//   ("for", "the"),
```

```
// ...
```

```
val numDimensions = frequentWords.size + topBigrams.size
// numDimensions: Int = 403
```

```
import axle.syntax.talliable.talliableOps

def featureExtractor(fp: Article): List[Double] = {

  val tokens = English.tokenize(fp.text.toLowerCase)
  val wordCounts = tokens.tally[Long]
  val bigramCounts =  bigrams(tokens).tally[Long]
  val wordFeatures = frequentWords.map(wordCounts(_) + 0.1)
  val bigramFeatures = topBigrams.map(bigramCounts(_) + 0.1)
  wordFeatures ++ bigramFeatures
}
```

Place a `MetricSpace` implicitly in scope that defines the space in which to measure similarity of Articles.

```
import spire.algebra._

import axle.algebra.distance.Euclidean

import org.jblas.DoubleMatrix
import axle.jblas.linearAlgebraDoubleMatrix

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
implicit val nrootDouble: NRoot[Double] = spire.implicits.DoubleAlgebra

implicit val space = {
  implicit val ringInt: Ring[Int] = spire.implicits.IntAlgebra
  implicit val inner = axle.jblas.rowVectorInnerProductSpace[Int, Int, Double]
(numDimensions)
  new Euclidean[DoubleMatrix, Double]
}
```

Create 4 clusters using k-Means

```
import axle.ml.KMeans
import axle.ml.PCAFeatureNormalizer
```

```
import cats.implicits._
import spire.random.Generator.rng

val normalizer = (PCAFeatureNormalizer[DoubleMatrix] _).curried.apply(0.98)

val classifier = KMeans[Article, List, DoubleMatrix](
  articles,
  N = numDimensions,
  featureExtractor,
  normalizer,
  K = 4,
  iterations = 100)(rng)
```

Show cluster vs author in a confusion matrix:

```
import axle.ml.ConfusionMatrix

val confusion = ConfusionMatrix[Article, Int, String, Vector, DoubleMatrix](
  classifier,
  articles.toVector,
  _.author,
  0 to 3)
```

```
confusion.show
// res11: String = """21 19  8  4 : 52 HAMILTON
//  3  0  0  0 :  3 HAMILTON AND MADISON
// 10  0  5  0 : 15 MADISON
//  2  0  0  3 :  5 JAY
// 10  0  0  1 : 11 HAMILTON OR MADISON
//
// 46 19 13  8
// """
```

Genetic Algorithms

See the wikipedia page on **Genetic Algorithms**

Example: Rabbits

Consider a `Rabbit` class

```
case class Rabbit(a: Int, b: Double, c: Double, d: Double, e: Double, f: Double, g: Double, h:
```

Define the `Species` for a Genetic Algorithm, which requires a random generator and a fitness function.

```
import shapeless._
```

```
val gen = Generic[Rabbit]

import axle.ml._

import scala.util.Random.nextDouble
import scala.util.Random.nextInt

implicit val rabbitSpecies = new Species[gen.Repr] {

  def random(rg: spire.random.Generator): gen.Repr = {

    val rabbit = Rabbit(
      1 + nextInt(2),
      5 + 20 * nextDouble(),
      1 + 4 * nextDouble(),
      3 + 10 * nextDouble(),
      10 + 5 * nextDouble(),
      2 + 2 * nextDouble(),
      3 + 5 * nextDouble(),
      2 + 10 * nextDouble())
    gen.to(rabbit)
  }

  def fitness(rg: gen.Repr): Double = {
    val rabbit = gen.from(rg)
    import rabbit._
    a * 100 + 100.0 * b + 2.2 * (1.1 * c + 0.3 * d) + 1.3 * (1.4 * e - 3.1 * f
 + 1.3 * g) - 1.4 * h
  }

}
```

Run the genetic algorithm

```
import cats.implicits._

val ga = GeneticAlgorithm(populationSize = 100, numGenerations = 100)

val log = ga.run(spire.random.Generator.rng)
```

```
val winner = log.winners.last
// winner: gen.Repr = 2 :: 24.852843762333084 :: 4.588011179322346 ::
 12.775841572444314 :: 14.70778125869996 :: 2.0249572093621238 ::
 7.974308212270284 :: 2.08110277512107 :: HNil
```

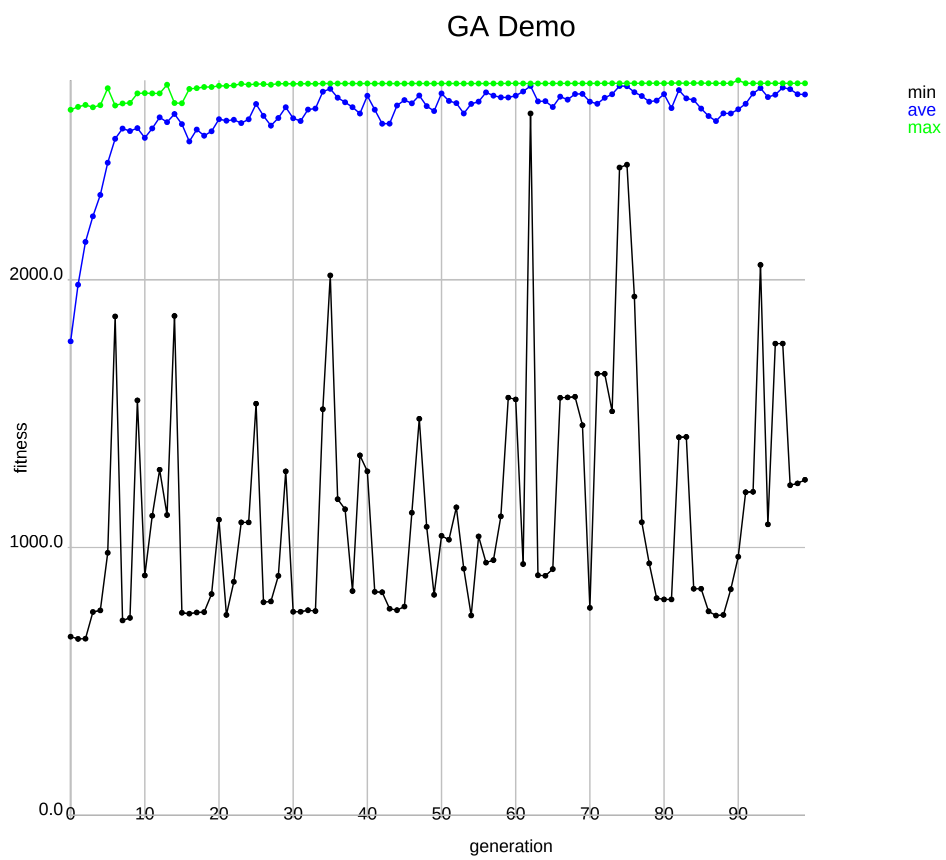
Plot the min, average, and max fitness function by generation

```
import scala.collection.immutable.TreeMap
import axle.visualize._
```

```
val plot = Plot[String, Int, Double, TreeMap[Int,Double]](
  () => List("min" -> log.mins, "ave" -> log.aves, "max" -> log.maxs),
  connect = true,
  colorOf = (label: String) => label match {
    case "min" => Color.black
    case "ave" => Color.blue
    case "max" => Color.green },
  title = Some("GA Demo"),
  xAxis = Some(0d),
  xAxisLabel = Some("generation"),
  yAxis = Some(0),
  yAxisLabel = Some("fitness"))
```

Render to an SVG file

```
import axle.web._
import cats.effect._

plot.svg[IO]("docwork/images/ga.svg").unsafeRunSync()
```



Biology

Needleman-Wunsch

See the Wikipedia page on the **Needleman-Wunsch** algorithm.

Example

Imports and implicits

```
import org.jblas.DoubleMatrix

import cats.implicits._

import spire.algebra.Ring
import spire.algebra.NRoot
import spire.algebra.Field

import axle.algebra._
import axle.algebra.functors._
import axle.bio._
import NeedlemanWunsch.optimalAlignment
import NeedlemanWunschDefaults._

implicit val nrootDouble: NRoot[Double] = spire.implicits.DoubleAlgebra
implicit val ringInt: Ring[Int] = spire.implicits.IntAlgebra
import axle.algebra.modules.doubleIntModule

implicit val laJblasInt = {
  implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
  axle.jblas.linearAlgebraDoubleMatrix[Double]
}
```

```
val dna1 = "ATGCGGCC"
val dna2 = "ATCGCCGG"
```

Setup

```
val nwAlignment = optimalAlignment[IndexedSeq, Char, DoubleMatrix, Int, Double]
(
  dna1, dna2, similarity, gap, gapPenalty)
// nwAlignment: (IndexedSeq[Char], IndexedSeq[Char]) = (
//   Vector('A', 'T', 'G', 'C', 'G', 'G', 'C', 'C', '-', '-'),
//   Vector('A', 'T', '-', 'C', '-', 'G', 'C', 'C', 'G', 'G')
// )
```

Score aligment

```
import NeedlemanWunsch.alignmentScore

alignmentScore(nwAlignment._1, nwAlignment._2, gap, similarity, gapPenalty)
// res0: Double = 32.0
```

Compute distance

```
val space
 = NeedlemanWunschSimilaritySpace[IndexedSeq, Char, DoubleMatrix, Int, Double]
(similarity, gapPenalty)
// space: NeedlemanWunschSimilaritySpace[IndexedSeq, Char, DoubleMatrix, Int,
 Double] = NeedlemanWunschSimilaritySpace(
//   baseSimilarity = <function2>,
//   gapPenalty = -5.0
// )

space.similarity(dna1, dna2)
// res1: Double = 32.0
```

Smith-Waterman

See the Wikipedia page on the **Smith-Waterman** algorithm.

Smith-Waterman Example

Imports and implicits

```
import org.jblas.DoubleMatrix

import cats.implicits._

import spire.algebra.Ring
import spire.algebra.NRoot

import axle.bio._
import SmithWatermanDefaults._
import SmithWaterman.optimalAlignment

implicit val ringInt: Ring[Int] = spire.implicits.IntAlgebra
implicit val nrootInt: NRoot[Int] = spire.implicits.IntAlgebra
implicit val laJblasInt = axle.jblas.linearAlgebraDoubleMatrix[Int]
```

Setup

```
val dna3 = "ACACACTA"
val dna4 = "AGCACACA"
```

Align the sequences

```
val swAlignment = optimalAlignment[IndexedSeq, Char, DoubleMatrix, Int, Int](
  dna3, dna4, w, mismatchPenalty, gap)
// swAlignment: (IndexedSeq[Char], IndexedSeq[Char]) = (
//   Vector('A', '-', 'C', 'A', 'C', 'A', 'C', 'T', 'A'),
//   Vector('A', 'G', 'C', 'A', 'C', 'A', 'C', '-', 'A')
// )
```

Compute distance of the sequences

```
val space
 = SmithWatermanSimilaritySpace[IndexedSeq, Char, DoubleMatrix, Int, Int]
(w, mismatchPenalty)
// space: SmithWatermanSimilaritySpace[IndexedSeq, Char, DoubleMatrix, Int,
 Int] = SmithWatermanSimilaritySpace(
//   w = <function3>,
//   mismatchPenalty = -1
// )

space.similarity(dna3, dna4)
// res3: Int = 12
```

Text

Natural Langage Processing (NLP), Linguistics, and Programming Languages

Language Modules

Natural-language-specific stop words, tokenization, stemming, etc.

English

Currently English is the only language module. A language modules supports tokenization, stemming, and stop words. The stemmer is from tartarus.org, which is released under a compatible BSD license. (It is not yet available via Maven, so its source has been checked into the Axle github repo.)

Example

```
val text = """
Now we are engaged in a great civil war, testing whether that nation, or any
 nation,
so conceived and so dedicated, can long endure. We are met on a great battle-
field of
that war. We have come to dedicate a portion of that field, as a final resting
 place
for those who here gave their lives that that nation might live. It is
 altogether
fitting and proper that we should do this.
"""
```

Usage

```
import axle.nlp.language.English

English.
  tokenize(text.toLowerCase).
  filterNot(English.stopWords.contains).
  map(English.stem).
  mkString(" ")
// res0: String = "now we engag great civil war test whether nation ani nation
 so conceiv so dedic can long endur we met great battle-field war we have come
 dedic portion field final rest place those who here gave live nation might
 live altogeth fit proper we should do"
```

Edit Distance

See the Wikipedia page on **Edit distance**

Levenshtein

See the Wikipedia page on **Levenshtein distance**

Imports and implicits

```
import org.jblas.DoubleMatrix

import cats.implicits._

import spire.algebra.Ring
import spire.algebra.NRoot

import axle._
import axle.nlp.Levenshtein
import axle.jblas._

implicit val ringInt: Ring[Int] = spire.implicits.IntAlgebra
implicit val nrootInt: NRoot[Int] = spire.implicits.IntAlgebra
implicit val laJblasInt = linearAlgebraDoubleMatrix[Int]
implicit val space = Levenshtein[IndexedSeq, Char, DoubleMatrix, Int]()
```

Usage

```
space.distance("the quick brown fox", "the quik brown fax")
// res2: Int = 2
```

Usage with spire's `distance` operator

Imports

```
import axle.algebra.metricspaces.wrappedStringSpace
import spire.syntax.metricSpace.metricSpaceOps
```

Usage

```
"the quick brown fox" distance "the quik brown fax"
// res3: Int = 2

"the quick brown fox" distance "the quik brown fox"
// res4: Int = 1

"the quick brown fox" distance "the quick brown fox"
// res5: Int = 0
```

Vector Space Model

See the Wikipedia page on **Vector space model**

Example

```
val corpus = Vector(
    "a tall drink of water",
    "the tall dog drinks the water",
    "a quick brown fox jumps the other fox",
    "the lazy dog drinks",
    "the quick brown fox jumps over the lazy dog",
    "the fox and the dog are tall",
    "a fox and a dog are tall",
    "lorem ipsum dolor sit amet"
)
```

Unweighted Distance

The simplest application of the vector space model to documents is the unweighted space:

```
import cats.implicits._

import spire.algebra.Field
import spire.algebra.NRoot

import axle.nlp.language.English
import axle.nlp.TermVectorizer

implicit val fieldDouble: Field[Double] = spire.implicits.DoubleAlgebra
implicit val nrootDouble: NRoot[Double] = spire.implicits.DoubleAlgebra

val vectorizer = TermVectorizer[Double](English.stopWords)
```

```
val v1 = vectorizer(corpus(1))
// v1: Map[String, Double] = Map(
//   "tall" -> 1.0,
//   "dog" -> 1.0,
//   "drinks" -> 1.0,
//   "water" -> 1.0
// )

val v2 = vectorizer(corpus(2))
// v2: Map[String, Double] = Map(
//   "brown" -> 1.0,
//   "quick" -> 1.0,
//   "jumps" -> 1.0,
//   "fox" -> 2.0,
//   "other" -> 1.0
// )
```

The object defines a `space` method, which returns a `spire.algebra.MetricSpace` for document vectors:

```
import axle.nlp.UnweightedDocumentVectorSpace

implicit val unweighted = UnweightedDocumentVectorSpace().normed
```

```
unweighted.distance(v1, v2)
// res7: Double = 3.4641016151377544

unweighted.distance(v1, v1)
// res8: Double = 0.0
```

Compute a "distance matrix" for a given set of vectors using the metric space:

```
import axle.jblas._
import axle.algebra.DistanceMatrix

val dm = DistanceMatrix(corpus.map(vectorizer))
```

```
dm.distanceMatrix.show
// res9: String = """0.000000 1.732051 3.316625 2.449490 3.162278 2.000000
 2.000000 2.828427
// 1.732051 0.000000 3.464102 1.732051 3.000000 1.732051 1.732051 3.000000
// 3.316625 3.464102 0.000000 3.316625 2.236068 2.645751 2.645751 3.605551
// 2.449490 1.732051 3.316625 0.000000 2.449490 2.000000 2.000000 2.828427
// 3.162278 3.000000 2.236068 2.449490 0.000000 2.449490 2.449490 3.464102
// 2.000000 1.732051 2.645751 2.000000 2.449490 0.000000 0.000000 2.828427
// 2.000000 1.732051 2.645751 2.000000 2.449490 0.000000 0.000000 2.828427
// 2.828427 3.000000 3.605551 2.828427 3.464102 2.828427 2.828427 0.000000"""

dm.distanceMatrix.max
// res10: Double = 3.605551275463989
```

TF-IDF Distance

```
import axle.nlp.TFIDFDocumentVectorSpace

val tfidf = TFIDFDocumentVectorSpace(corpus, vectorizer).normed
```

```
tfidf.distance(v1, v2)
// res11: Double = 4.068944074907273

tfidf.distance(v1, v1)
// res12: Double = 0.0
```

Angluin Learner

Models **Dana Angluin**'s Language Learner.

Example: Baby Angluin Learner

Imports

```
import axle._
import axle.lx._
import Angluin._
```

Setup

```
val mHi = Symbol("hi")
val mIm = Symbol("I'm")
val mYour = Symbol("your")
val mMother = Symbol("Mother")
val mShut = Symbol("shut")
val mUp = Symbol("up")

val Σ = Alphabet(Set(mHi, mIm, mYour, mMother, mShut, mUp))

val s1 = Expression(mHi :: mIm :: mYour :: mMother :: Nil)
val s2 = Expression(mShut :: mUp :: Nil)
val # = Language(Set(s1, s2))

val T = Text(s1 :: # :: # :: s2 :: # :: s2 :: s2 :: Nil)

val # = memorizingLearner
```

Usage

```
import axle.algebra.lastOption

val outcome = lastOption(#.guesses(T))
// outcome: Option[Grammar] = Some(
//   value = HardCodedGrammar(
//     # = Language(
//       sequences = Set(
//         Expression(symbols = List('hi, 'I'm, 'your, 'Mother)),
//         Expression(symbols = List('shut, 'up))
//       )
//     )
//   )
// )

outcome.get.#
// res14: Language = Language(
//   sequences = Set(
```

```
//      Expression(symbols = List('hi, 'I'm, 'your, 'Mother)),
//      Expression(symbols = List('shut, 'up))
//    )
// )

#
// res15: Language = Language(
//    sequences = Set(
//      Expression(symbols = List('hi, 'I'm, 'your, 'Mother)),
//      Expression(symbols = List('shut, 'up))
//    )
// )

T
// res16: Text = Iterable(
//    Expression(symbols = List('hi, 'I'm, 'your, 'Mother)),
//    Expression(symbols = List()),
//    Expression(symbols = List()),
//    Expression(symbols = List('shut, 'up)),
//    Expression(symbols = List()),
//    Expression(symbols = List('shut, 'up)),
//    Expression(symbols = List('shut, 'up))
// )

T.isFor(#)
// res17: Boolean = true
```

Gold Paradigm

Models the Gold Paradigm.

Example: Baby Gold Learner

Imports

```
import axle._
import axle.lx._
import GoldParadigm._
```

Setup

```
val mHi = Morpheme("hi")
val mIm = Morpheme("I'm")
val mYour = Morpheme("your")
val mMother = Morpheme("Mother")
val mShut = Morpheme("shut")
val mUp = Morpheme("up")

val Σ = Vocabulary(Set(mHi, mIm, mYour, mMother, mShut, mUp))
```

```
val s1 = Expression(mHi :: mIm :: mYour :: mMother :: Nil)
val s2 = Expression(mShut :: mUp :: Nil)

val ℒ = Language(Set(s1, s2))

val T = Text(s1 :: ℒ :: ℒ :: s2 :: ℒ :: s2 :: s2 :: Nil)

val ℒ = memorizingLearner
```

Usage

```
import axle.algebra.lastOption

lastOption(ℒ.guesses(T)).get
// res19: Grammar = HardCodedGrammar(
//   ℒ = Language(
//     sequences = Set(
//       Expression(
//         morphemes = List(
//           Morpheme(s = "hi"),
//           Morpheme(s = "I'm"),
//           Morpheme(s = "your"),
//           Morpheme(s = "Mother")
//         )
//       ),
//       Expression(morphemes = List(Morpheme(s = "shut"), Morpheme(s = "up")))
//     )
//   )
// )

ℒ
// res20: Language = Language(
//   sequences = Set(
//     Expression(
//       morphemes = List(
//         Morpheme(s = "hi"),
//         Morpheme(s = "I'm"),
//         Morpheme(s = "your"),
//         Morpheme(s = "Mother")
//       )
//     ),
//     Expression(morphemes = List(Morpheme(s = "shut"), Morpheme(s = "up")))
//   )
// )

T
// res21: Text = Text(
//   expressions = List(
//     Expression(
//       morphemes = List(
//         Morpheme(s = "hi"),
//         Morpheme(s = "I'm"),
```

```
//          Morpheme(s = "your"),
//          Morpheme(s = "Mother")
//        )
//      ),
//      Expression(morphemes = List()),
//      Expression(morphemes = List()),
//      Expression(morphemes = List(Morpheme(s = "shut"), Morpheme(s = "up"))),
//      Expression(morphemes = List()),
//      Expression(morphemes = List(Morpheme(s = "shut"), Morpheme(s = "up"))),
//      Expression(morphemes = List(Morpheme(s = "shut"), Morpheme(s = "up")))
//    )
// )

T.isFor(#)
// res22: Boolean = true
```

Python Grammar

This is part of a larger project on source code search algorithms.

`python2json.py` will take any python 2.6 (or older) file and return a json document that represents the abstract syntax tree. There are a couple of minor problems with it, but for the most part it works.

As an example, let's say we have the following python in example.py:

```
x = 1 + 2
print x
```

Invoke the script like so to turn example.py into json:

```
python2json.py -f example.py
```

You can also provide the input via stdin:

```
cat example.py | python2json.py
```

I find it useful to chain this pretty-printer when debugging:

```
cat example.py | python2json.py | python -mjson.tool
```

The pretty-printed result in this case is:

```
{
    "_lineno": null,
    "node": {
        "_lineno": null,
        "spread": [
```

```
            {
                "_lineno": 2,
                "expr": {
                    "_lineno": 2,
                    "left": {
                        "_lineno": 2,
                        "type": "Const",
                        "value": "1"
                    },
                    "right": {
                        "_lineno": 2,
                        "type": "Const",
                        "value": "2"
                    },
                    "type": "Add"
                },
                "nodes": [
                    {
                        "_lineno": 2,
                        "name": "x",
                        "type": "AssName"
                    }
                ],
                "type": "Assign"
            },
            {
                "_lineno": 3,
                "nodes": [
                    {
                        "_lineno": 3,
                        "name": "x",
                        "type": "Name"
                    }
                ],
                "type": "Println"
            }
        ],
        "type": "Stmt"
    },
    "type": "Module"
}
```

Future Work

Python Grammar organization

- factor out `axle-ast-python`
- `axle-ast-python`

AST

- move ast view xml (how is it able to refer to `xml.Node`?)

- ast.view.AstNodeFormatter (xml.Utility.escape)

- ast.view.AstNodeFormatterXhtmlLines

- ast.view.AstNodeFormatterXhtml
- Tests for `axle.ast`
- Redo axle.ast.* (rm throws, more typesafe)
- `cats.effect` for `axle.ast.python2`

Linguistics

- Nerod Partition
- Finish Angluin Learner
- Motivation for Gold Paradigm, Angluin Learner

Quantum Circuits

To be written...

Future Work

Soon

- `QBit2.factor`
- Fix and enable `DeutschOracleSpec`
- QBit CCNot

Later

- Shor's algorithm
- Property test reversibility (& own inverse)
- Typeclass for "negate" (etc), Binary, CBit
- Typeclass for unindex
- Deutsch-Jozsa algorithm (D.O. for n-bits) (Oracle separation between EQP and P)
- Simon's periodicity problem (oracle separation between BQP and BPP)
- Grover's algorithm
- Quantum cryptographic key exchange

Appendix

Road Map

See **Release Notes** for the record of previously released features.

0.6.7

- Near-term stuff from **quantum circuit future work**

0.7.x Scala 3

See Scala 3 section of **future work** for foundation

0.8.x Game

See **Future Work** for Axle Game

0.9.x Randomness and Uncertainty

Factoring and Bayesian Networks

See **Future Work** for Randomness and Uncertainty

0.10.x Bugs and adoption barriers

See **Future work** for Foundation

0.11.x Text improvements

- Near-term stuff from **text**

0.12.x Visualization

See **future work** for axle visualization

0.13.x Mathematics

See **future work**

Build and Deploy

For contributors

- **Source code** on GitHub

- Build status on Github Actions CI Release passing
- Code coverage codecov 74%

Publish snapshots

Push commits to repo.

Monitor **progress** of github action.

Confirm jars are present at the **sonatype snapshot repo**

Release new version

For example, tag with a version:

```
git tag -a v0.1.6 -m "v.0.1.6"
git push origin v0.1.6
```

Monitor **progress**

Confirm jars are present at the **sonatype repo**

Update Site

Run the `site-update.sh` script

Monitor **progress** of action.

Verify by browsing to the **site** or look at the `gh-pages` **branch**

Verify before update

Just to do the build locally, run

```
sbt -J-Xmx8G 'project axle-docs' mdoc
sbt 'project axle-docs' laikaSite
```

To preview the changes, do:

```
sbt 'project axle-docs' laikaPreview
```

then browse to **https://localhost:4242**

If it looks good, push with:

```
sbt 'project axle-docs' ghpagesCleanSite ghpagesPushSite
```

Monitor and verify as before.

References for Build and Deploy

- Laika

- http4s Laika PR
- sbt-site
- sbt-ghpages

- Note the instructions to set up a `gh-pages` branch
- custom domain for github pages

- Note instructions for apex domains
- sbt-sonatype

- sonatype using credentials in `~/.sbt/1.0/sonatype.sbt`
- sbt-ci-release

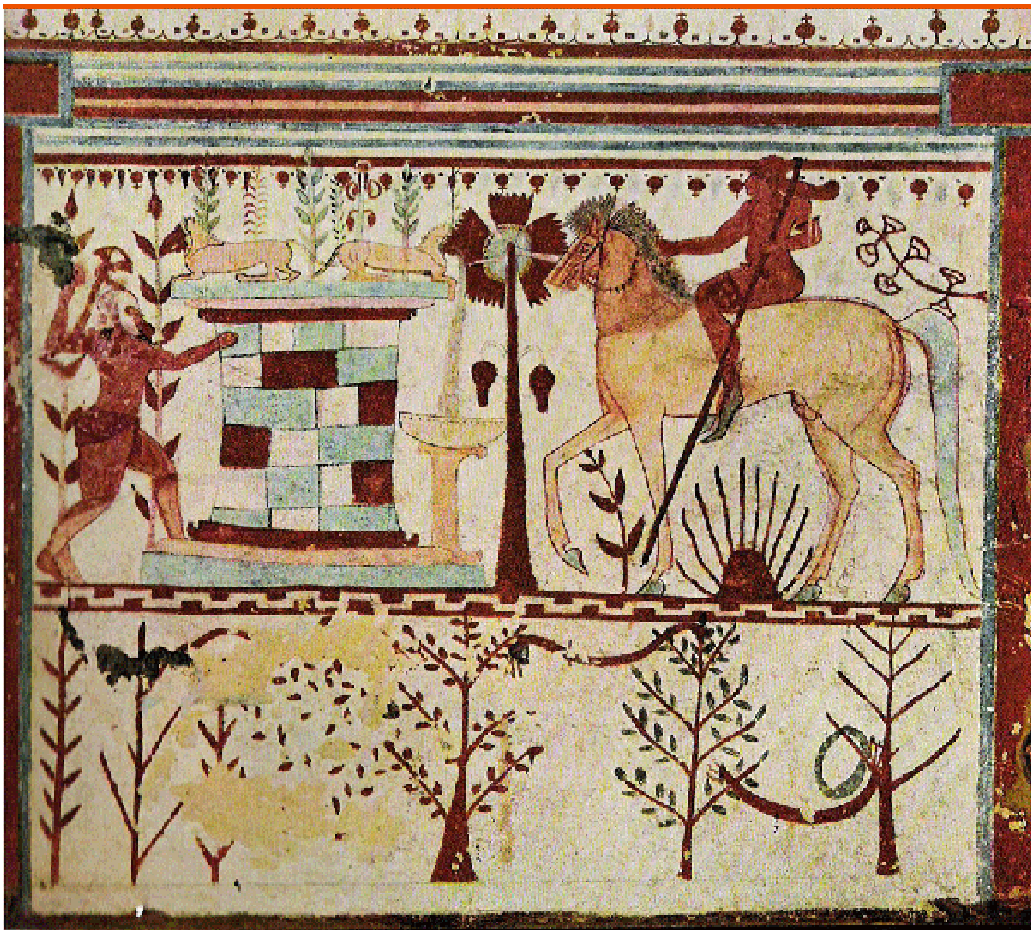
History

####

Axle models a set of formal subjects that the author has encountered throughout his lifetime. They take the form of functioning code that allows the reader to experiment with alternative examples.

Although the primary aim of this code is education and clarity, scalability and performance are secondary goals.

The name "axle" was originally chosen because it sounds like "Haskell". Given the use of UTF symbols, I tried spelling it using Greek letters to get "####". It turns out that this is the Etruscan spelling of Achilles



(**image context**)

Follow **@axledsl** on Twitter.

Project History References

Quanta

The first time I had the idea to group units into quanta was at NOCpulse (2000-2002). NOCpulse was bought by Red Hat, which open-sourced the code. There is still **evidence** of that early code online.

In a 2006 class given by **Alan Kay** at UCLA, I proposed a system for exploring and learning about scale. The idea occurred to me after reading a news article about a new rocket engine that used the Hoover Dam as a point of reference. I wound up implementing another idea, but always meant to come back to it.

Machine Learning

Based on many classes at Stanford (in the 90's) and UCLA (in the 00's), and more recently the Coursera machine learning course in the Fall of 2011. The inimitable **Artificial Intelligence: A Modern Approach** has been a mainstay throughout.

Statistics, Information Theory, Bayesian Networks, & Causality

The Information Theory code is based on **Thomas Cover**'s **Elements of Information Theory** and his EE 376A course.

I implemented some Bayesian Networks code in Java around 2006 while **Adnan Darwiche** class on the subject at UCLA. The Axle version is based on his book, **Modeling and Reasoning with Bayesian Networks**

Similarly, I implemented ideas from **Judea Pearl** UCLA course on Causality in Java. The Axle version is based on his classic text **Causality**

Game Theory

As a senior CS major at Stanford in 1996, I did some independent research with Professor **Daphne Koller** and PhD student **Avi Pfeffer**.

This work spanned two quarters. The first quarter involved using Koller and Pfeffer's **Gala** language (a Prolog-based DSL for describing games) to study a small version of Poker and solve for the **Nash equilibria**. The second (still unfinished) piece was to extend the solver to handle non-zero-sum games.

The text I was using at the time was **Eric Rasmusen**'s **Games and Information**

Linguistics

Based on notes from **Ed Stabler**'s graduate courses on language evolution and computational linguistics (Lx 212 08) at UCLA.

Author

Adam Pingel is an Iowa native who wrote his first lines of code on an Apple][in 1983.

He moved to the San Francisco Bay Area in 1992 to study Computer Science at Stanford. After graduating in 1996, spent several years at Excite.com, helping it scale to become the 3rd largest site on the web at the time. After Excite he spent two years at NOCpulse -- a startup acquired by Red Hat.

In 2002 he left Silicon Valley to join the UCLA Computer Science department's PhD program. His major field was programming languages and systems, and his minor fields were AI and Linguistics. His first year he worked as a TA for the undergraduate Artificial Intelligence class. The second was spent as a graduate student researcher working on programming tools for artists at the Hypermedia Lab (a part of the UCLA School of Theater, Film, and Television). From 2005 - 2009 he mixed graduate studies with consulting. He received an MS along the way, and ultimately decided to pursue his research interests in the open source community.

In April 2009 he moved to San Francisco and joined the Independent Online Distribution Alliance (IODA) as the Lead Systems Engineer. During his time there, IODA was acquired by Sony Music and then The Orchard. In April 2012 he co-founded Eddgy. In May 2013 he joined VigLink as Staff Software Engineer and later managed a team there. In September 2015 he became VP of Engineering at Ravel Law. In June 2017 Ravel Law was acquired by LexisNexis, where he became a Sr. Director. In late 2019, he became the CTO of Global Platforms. In early 2022, he joined IBM's **Accelerated Science** team as a technical lead.

For more background, see his accounts on:

- **LinkedIn**
- **StackOverflow**
- **@pingel** on Twitter
- **Google Scholar**

He can be reached at `adam@axle-lang.org`.

Videos

"Axle" talk at Scala by the Bay 2015

"Lawful AI" talk at Scale by the Bay 2017

Release Notes

See **Road Map** for the plan of upcoming releases and features.

0.6.x

0.6.1-6 (April-May 2022)

- CI/CD and Site enhancements
- Automated releases via `sbt-ci-release`
- Move away from PRs
- Adopt Laika for site and PDF generation

0.6.0 cats.effect for axle.game (December 31, 2020)

- Wrap `axle.IO.getLine` in `F[_]`
- Remove from `Game`: method `probabilityDist`, `sampler`, and type params `V` and `PM[_, _]`
- Move `strategyFor` from `Game` to `strategies` argument in `axle.game` package methods

- Define `Indexed.slyce` for non-1-step Ranges
- Improve `axle.lx.{Gold, Angluin}` coverage
- `axle.laws.generator` includes generators for GeoCoordinates, UnittedQuantities, and Units
- Simpler `hardCodedStrategy` and `aiMover` signatures
- Replace `randomMove` with `ConditionalProbabilityTable.uniform`

0.5.x

0.5.4 Sampler Axioms + package reorg (September 28, 2020)

- Sampler Axioms
  1. ProbabilityOf(RegionEq(sample(gen))) > 0?
  2. Sampled distribution converges to model's
- Pre-compute `ConditionalProbabilityTable.bars` for `Sampler` witness
- Move everything from `axle._` into sub-packages (`algebra`, `math`, `logic`)
- Organize `axle.algebra._` package object
- `axle.laws.generator`
- `rationalProbabilityDist` is now implicitly available

0.5.3 (September 13, 2020)

- Split `ProbabilityModel` into three new typeclasses -- `Bayes`, `Kolmogorov`, `Sampler` -- as well as `cats.Monad`. The three axle typeclasses include syntax.
- Rename `ConditionalProbabilityTable.values` to `domain`
- Bugs fixed

- Bayes axiom should avoid P(A) == P(B) == 0

- `UnittedQuantity` `LengthSpace` unit mismatch

- `BarChart` was missing `Order[C]`
- Expanded documentation

0.5.2 (September 7, 2020)

- Move to Scala 2.12 and 2.13

- Changes in `axle.game` to provide `Generator` where needed, and return a `ConditionalProbabilityTable0`
- Redo `axle.stats`
- `ProbabilityModel` typeclass (refactored from `Distribution`) including syntactic support
- Implicitly conjurable `cats.Monad` from a `ProbabilityModel`, which supports for comprehensions via cats syntax support
- `Variable` instead of `RandomVariable`
- remove `Bayes`
- `axle.quantumcircuit` package for modelling computing with quantum circuits
- Replace `axle.agebra.Zero` with `spire.algebra.AdditiveMonoid.zero`
- Remove `axle-spark` (Spark "spoke") for now
- Move `axle.ml.distance` to `axle.algebra.distance`
- `axle.dummy` for a handful of scanLeft calls
- Remove Spark impacts on typeclasses in `axle.algebra`. Eg: Spark's `ClassTag` requirement `map` created the difficulty:
- `Functor`: removed and replaced with `cats.Functor`
- `Scanner`, `Aggregator`, `Zipper`, `Indexed`, `Talliable`, `Finite`: Refactored as Kind-1 typeclasses
- Vertex and Edge projections for jung graphs
- Fix `axle.joda.TicsSpec` handling of timezones
- ScaleExp works with negative exponent
- ScalaCheck tests for
- Group and Module of UnittedQuantity
- MetricSpace axle.algebra.GeoMetricSpace
- `axle.ml.GeneticAlgorithm` rewritten in terms of **kittens**
- `Show`, `Order`, `Eq` witnesses
- Eq.fromUniversalEquals where applicable

- SAM inference elsewhere
- Remove `axle.string` and `axle.show`.
- Replace uses with `.show` from `cats.implicits` or show string interpolation
- Remove extraneous `cutoff` argument for `PCA`
- Replace Tut with MDoc
- Lawful ScalaCheck tests for
- `Modules` in `axle.algebra`
- `SimilaritySpaces` for `SmithWaterman` & `NeedlemanWunsch`
- Fix `Order[Card]`
- `Deck.riffleShuffle`
- `GuessRiffle` game
- `axle.algebra.etc` via `axle.algebra.EnrichedRinged`
- `bernoulliDistribution`
- `axle.stats.expectation(CPT)`
- `axle.IO` consolidates IO to `cats.effect` (eg `[F[_]: ContextShift: Sync]`)
- Create `axle-awt`, `axle-xml`, and `axle-jogl` (leaving `axle.scene.{Shape,Color}` in `axle-core`)
- Remove `axle-jogl` due to instability of underlying dependencies

0.4.x

0.4.1 (June 4, 2017)

- Fix all warnings, and turn on fatal warnings
- DrawPanel typeclass
- Configurable visualization parameters for {un,}directedGraph and BayesianNetwork
- Make Monix "provided"

0.4.0 (May 30, 2017)

- axle-core gets axle-visualize and most of axle-algorithm
- new axle-wheel formed from axle-{test, games, languages} and parts of axle-algorithms

0.3.x

0.3.6 (May 29, 2017)

- Replace Akka with Monix for animating visualizations
- `ScatterPlot` `play` to awt

0.3.5 (May 23, 2017)

- Move math methods from `axle.algebra._` package object to `axle.math._`

0.3.4 (May 22, 2017)

- Move mathy methods from `axle._` package object to new `axle.math._` package object
- Sieve of Eratosthenes
- Remove some `Eq` and `Order` witnesses from `axle._` as they are now available in `cats._`
- Revert Tut to version 0.4.8

0.3.3 (May 7, 2017)

- `BarChart.hoverof` -- center text in bar
- `BarChart{,Grouped}.linkOf`

0.3.2 (May 6, 2017)

- Remove `axle.jblas.{additiveCMonoidDoubleMatrix, multiplicativeMonoidDoubleMatrix, module, ring}
- `axle.math.exponentiateByRecursiveSquaring`
- Rename `fibonacci*` methods
- `PixelatedColoredArea` should take a function that is given a rectangle (not just a point)
- Logistic Map vis using `PixelatedColoredArea` (documentation)

0.3.1 (May 1, 2017)

- `BarChart*.hoverOf`
- `BarChart*` label angle is Option. None indicates no labels below bars.
- `axle.xml` package in axle-visualize

0.3.0 (April 12, 2017)

- Scala org to Typelevel
- Fix malformed distribution in ConditionalProbabilityTable and TallyDistribution0
- Depend on Spire 0.14.1 (fix mistaken dependency on snapshot release in 0.2.8)

0.2.x

0.2.8 (March 28, 2016)

- Fix SVG rendering of negative values in BarChart
- Make more arguments to vis components functions (colorOf, labelOf, diameterOf)
- Depend on Spire 0.13.1-SNAPSHOT (which depends on Typelevel Algebra)

0.2.7 (January 2016)

- Use cats-kernel's Eq and Order in favor of Spire's (with Shims to continue to work with Spire)
- Convert tests to use scalatest (to match Cats and Spire)

0.2.6 (November 2016)

- Depends on cats-core (initially just for Show typeclass)
- Strategy: `(G, MS) => Distribution[M, Rational]`
- LinearAlgebra.from{Column,Row}MajorArray
- Implementation of Monty Hall using axle.game typeclasses
- Implementaiton of Prisoner's Dilemma using axle.game typeclasses
- Minor Poker fixes

0.2.5 (October 2016)

- Typeclasses for axle.game
- Increase test coverage to 78%

0.2.4 (September 5, 2016)

- Redo all and extend documentation using Tut
- Convert Build.scala to build.sbt
- LinearAlgebra doc fixes / clarification

- Make some axle.nlp.Corpus methods more consistent
- Avoid using wget in axle.data._
- float*Module witnesses in axle._

0.2.3 (July 30, 2016)

- ScatterPlot
- Logistic Map and Mandelbrot
- PixelatedColoredArea

0.2.2 (October 10, 2015)

- Pythagorean means

0.2.0 (August 12, 2015)

- reorganize to minimize dependencies from axle-core, with witnesses in the axle-X jars (axle.X package) for library X
- LinearAlgebra typeclass
- Functor, Aggregatable typeclasses
- Show, Draw, Play typeclasses
- MAP@k, harmonicMean
- axle-spark
- Apache 2.0 license

0.1.x

0.1.13 through 0.1.17 (October 12, 2014)

- Distribution as a Monad
- Spire 'Module' for axle.quanta

0.1-M12 (June 26, 2014)

- Upgrade to Scala 2.11.1
- Field context bound for classes in axle.stats and pgm
- axle.quanta conversions as Rational

0.1-M11 (February 26, 2014)

- REPL
- 3d visualizations using OpenGL (via jogl)
- More prevalent use of Spire typeclasses and number types

0.1-M10 (May 14, 2013)

- bug fixes in cards and poker
- api changes and bug fixes to visualizations required by hammer
- upgrade to akka 2.2-M3 and spire 0.4.0

0.1-M9 (April 7, 2013)

- DNA sequence alignment algorithms in `axle.bio`
- `axle.logic`
- multi-project build, rename axle to axle-core, and split out axle-visualize

0.1-M8 (March 11, 2013)

- Akka for streaming data updates to Plot and Chart
- Tartarus English stemmer
- Create `axle.nlp` package and move much of `axle.lx` there
- Move Bayesian Networks code to `axle.pgm`
- `axle.actor` for Akka-related code

0.1-M7 (February 19, 2013)

- Use `spire.math.Number` in `axle.quanta`
- Use `spire.algebra.MetricSpace` for `axle.lx.*VectorSpace` and `axle.ml.distance.*`

0.1-M6 (February 13, 2013)

- Initial version of `axle.algebra`
- No mutable state (except for permutations, combinations, and mutable buffer enrichment)
- `axle.quanta` conversion graph edges as functions
- Redoing `JblasMatrixFactory` as `JblasMatrixModule` (preparing for "cake" pattern")

0.1-M5 (January 1, 2013)

- Bar Chart
- Minimax
- Texas Hold Em Poker

0.1-M4 (December 16, 2013)

- Clean up `axle.graph` by scrapping attempt at family polymorphism
- Generalize InfoPlottable to QuantaPlottable

0.1-M3 (December 11, 2012)

- Immutable graphs

0.1.M2 (October 24, 2012)

- Genetic Algorithms
- Bug: x and y axis outside of plot area
- Naive Bayes
- `show()` in `axle.visualize`
- PCA
- Immutable matrices
- Optimize Plot of `axle.quanta`

0.1.M1 (July 15, 2012)

- Jblas-backed Matrix
- Jung-backed Graph
- Quanta (units of measurement)
- Linear Regression
- K-means