Hyperspectral Image Analysis with a Functional Data Model

Doug Lindholm
Laboratory for Atmospheric and Space Physics
University of Colorado Boulder

UCAR Software Engineering Assembly
April 2, 2018

Outline

- The NASA HyLaTiS Project
- (Re)Introduction of the Functional Data Model
- Thinking about Data Functionally
- Modeling Data with the Functional Data Model
- Manipulating Data with Functional Algebra

HyLaTiS

- NASA funded project to support interactive analysis of hyperspectral imagery data
  - geo-referenced “grid” of spectra
- Big data problem (too big to fit into memory of a single machine)
- Distributed cloud-based solution (Spark)
- Functional Programming techniques
- Functional Data Model (LaTiS)

Functional Data Model

- Conceptual model that captures the functional relationship between independent and dependent variables
- Reference implementation in LaTiS – Open Source Scala library being developed at LASP
  - Version 2: https://github.com/latis-data/latis
  - Starting version 3
- But first…

Relational Data Model

- Relation = table
- Tuple = row
- Scalar = column

Only says that the values in a row **are** related but not **how**

```
(time, flux)
```

time	flux
0	1
1	2
2	3
3	2
4	1

Functional Data Model

time	flux
0	1
1	2
2	3
3	2
4	1

Constrains relations to *functions*

`flux(time)`

or

`time -> flux`

- Logically represents a sequence of *samples:* `(0,1), (1,2), (2,3), …`
- *Ordered* and *unique* domain values
- Given an interpolation strategy, a data function becomes a **computational** function that you can **evaluate:** `flux(1.5) => 2.5`

Arbitrary Complexity

Functional Data Model structures (*scalar*, *tuple*, *function*) can be nested

```
time -> (temperature, wind:(u, v))
```

Arbitrary Complexity

Functional Data Model structures (*scalar*, *tuple*, *function*) can be nested

```
time -> (temperature, wind:(u, v))
```

Time series of spectra

```
time -> wavelength -> flux
```

Arbitrary Complexity

Functional Data Model structures (*scalar*, *tuple*, *function*) can be nested

```
time -> (temperature, wind:(u, v))
```

Time series of spectra

```
time -> wavelength -> flux
```

Hyperspectral data cube (gridded spectra)

```
(lon, lat) -> wavelength -> flux
```

Array Equivalence

An array is just a function of index

```
i -> a
```

Array Equivalence

An array is just a function of index

```
i -> a
```

Separate arrays become one structure

```
time(i); flux(i)  =>  time -> flux
```

Array Equivalence

An array is just a function of index

```
i -> a
```

Separate arrays become one structure

```
time(i); flux(i)  =>  time -> flux
```

Akin to NetCDF coordinate variables

```
time(time); flux(time) =>  time -> flux
```

Array Equivalence

An array is just a function of index

```
i -> a
```

Separate arrays become one structure

```
time(i); flux(i)  =>  time -> flux
```

Akin to NetCDF coordinate variables

```
time(time); flux(time) =>  time -> flux
```

Multiple variables with shared dimensions becomes one structure using tuples

```
a(x); b(x); c(x)  =>  x -> (a, b, c)
```

Array Equivalence

An array is just a function of index

```
i -> a
```

Separate arrays become one structure

```
time(i); flux(i)  =>  time -> flux
```

Akin to NetCDF coordinate variables

```
time(time); flux(time) =>  time -> flux
```

Multiple variables with shared dimensions becomes one structure using tuples

```
a(x); b(x); c(x)  =>  x -> (a, b, c)
```

Multi-dimensional array

```
(x,y) -> (a,b,c)
```

Functional Algebra

Manipulating Data

- selection
- projection
- join
- groupBy
- pivot
- transpose
- curry
- partial application
- bijective functions
- function composition

Relational Algebra

The Functional Data Model inherits the mathematics of relational algebra

```
t -> (a,b)
```

t	a	b
10	1	on
20	2	off
30	3	on
40	2	on
50	1	off

Relational Algebra

The Functional Data Model inherits the mathematics of relational algebra

```
t -> (a,b)
```

Projection: reduce number of variables

- **project** `t,a  =>  t -> a`

t	a	b
10	1	on
20	2	off
30	3	on
40	2	on
50	1	off

Relational Algebra

The Functional Data Model inherits the mathematics of relational algebra

`t -> (a,b)`

Projection: reduce number of variables

- project t , a => `t -> a`
- **project** b => `i -> b`

  need index placeholder for domain

i	t	a	b
0	10	1	on
1	20	2	off
2	30	3	on
3	40	2	on
4	50	1	off

Relational Algebra

The Functional Data Model inherits the mathematics of relational algebra

```
t -> (a,b)
```

Projection: reduce number of variables

- project `t,a` => `t -> a`
- project `b` => `i -> b`

Selection: reduce number of samples

- **select** `t = 20`

t	a	b
10	1	on
20	**2**	**off**
30	3	on
40	2	on
50	1	off

Relational Algebra

The Functional Data Model inherits the mathematics of relational algebra

`t -> (a,b)`

Projection: reduce number of variables

- project `t`, `a` => `t -> a`
- project `b` => `i -> b`

Selection: reduce number of samples

- select `t = 20`
- **select** `a > 1`

t	a	b
10	1	on
20	**2**	**off**
30	**3**	**on**
40	**2**	**on**
50	1	off

Joins

Relational Algebra joins are based on column name.

Functional Algebra joins are based on domain sets.

```
t -> a + t -> b
t -> (a,b)
```

- Requires same domain type
- Interpolate if domain sets differ
- Otherwise nulls or fill values

Append time segments:

```
Jan:t -> a + Feb:t -> a + … => t -> a
```

Group By

Factor out a new domain

```
i -> (t, f)
groupBy(t)
t -> j -> f
```

Since `t` might not be unique, we may have multiple `f` values. Relational algebra requires aggregation (e.g. sum, mean)

If `t` is unique, we can reduce: `t -> f`

Pivot

Given a 2D Cartesian dataset

```
t -> w -> f
```

Make a column for each unique value of `w`

```
pivot(w)
t -> (f1, f2)
```

t	w	f
0	1	10
0	2	20
1	1	30
1	2	40

t	f1	f2
0	10	20
1	30	40

Transpose

```
(x,y) -> f
(y,x) -> f
```

- Implications for ordering (if only logical)

Currying

```
(t, w) -> f
t -> w -> f
```

- Requires Cartesian grid (same wavelengths for every time)
- Changes notion of “sample”

Partial Application

Given an uncurried spectral time series:

```
(t, w) -> f
```

we can *evaluate* it for a given time (`t`) and wavelength (`w`) to get a single flux (`f`) value.

Given a curried spectral time series:

```
t -> w -> f
```

we can *evaluate* it for a given time (`t`) and end up with a spectrum: `w -> f`

Bijective (one-to-one) Functions

Coordinate system transformations

```
(x, y) <-> (lon, lat)
```

Can go both ways.

Could be as simple as

```
index <-> time
```

For example, index into a data structure given a time value.

Function Composition

```
f: a -> b
g: b -> c
g.f: a -> c
```

Given that we can make a data function behave as a computational function given an interpolation strategy, we can compose algorithms with data!

HyLaTiS: Hyperspectral Imagery in the Cloud with Spark

HySICS: LASP built hyperspectral imager

Balloon Flight

flight direction (y)
slit orientation (x)

Slit Image

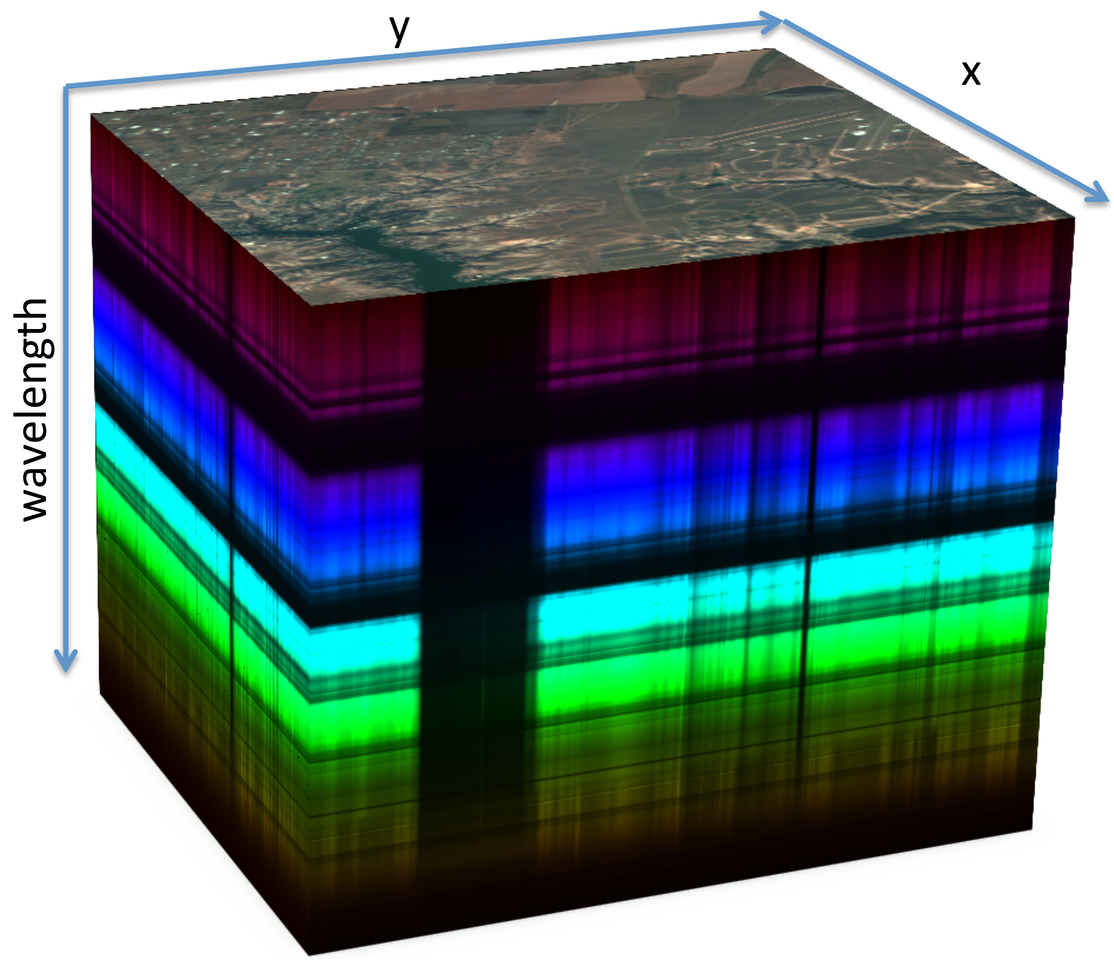
spectra at each pixel

`(x, w) -> f`

Data Cube

sequence of slit images

`y -> (x, w) -> f`



Algebraically Transform to Geo-Gridded Spectra

Original data cube: `y -> (x, w) -> f`

uncurry (3D array): `(y, x, w) -> f`

curry (gridded spectra): `(y, x) -> w -> f`

transpose: `(x, y) -> w -> f`

coordinate system transform (algorithm):

`(lon, lat) <-> (x, y)`

function composition (algorithm + data!):

`(lon, lat) -> w -> f`

Apache Spark

- Spark effectively provides a large virtual dataset that is distributed among nodes in a cluster, typically in memory.
- The trick is to partition your data to take advantage of parallelism while minimizing data shuffling.
- Resilient Distributed Dataset: RDD
- Effectively a collection of things of a given type “T”: RDD[T]

Spark and the Function Data Model

- Since a data function can be thought of as a sequence of Samples, we can define our Spark dataset as `RDD[Sample]`
- The functional algebra can generally be defined as computational functions of Samples: `f: Sample -> Sample`
- Spark can apply these operations to our distributed dataset (RDD) in parallel

Collecting an RGB image dataset

Start with geo-referenced data cube:

`(lon, lat) -> w -> f`

uncurry: `(lon, lat, w) -> f`

- sample = single voxel
- partioned far and wide

select `lon, lat` coverage

select three wavelength values for red, green, blue

pivot on wavelength

`(lon, lat) -> (red, green, blue)`

Partition Considerations
2D Data Compression

Start with geo-referenced data cube:

```
(lon, lat, w) -> f
```

groupBy `w`

```
w -> (lon, lat) -> f
```

- Causes data shuffling
- Partitioned by outer dimension: `w`
- each `w` slice is collocated so compression algorithm can work on each 2D grid

Summary

- Functional Data Model provides a useful abstraction for modeling and manipulating datasets
- Captures key elements of the Relational Data Model and Multi-dimensional array data model
- Can be applied to any dataset, logically or in practice.