Class 10: Metadata, Provenance & Test Thickets

Miguel F. Morales
Bryna Hazelton

Metadata

All the information about your data

- When it was taken
- What instrument took it
- How it was calibrated (calibration version, code that did calibration, etc.)
- Prior steps in analysis

Any information you need about the data for a plot

Metadata as a nutrition label for your data

Tagging your data with information

Metadata goals

- **Basic:** you can read the information about your data from the file (nutrition label for everything on a plot)

- **Goal:** you can recreate the analysis as needed (routines run, git hashes, etc.)

- **Advanced:** can recreate full instrument & analysis state (e.g. adds links back into Monitor & Control database; library versions)

How to store metadata

- ***Don't*** store it in the file name
  - Too mutable
  - Not enough space

- Almost all modern file formats have locations for metadata (e.g. headers)
  - FITS, hdf5, AVRO…

Using a standard binary file format (fits, hdf5, avro)

- Accurate (conversion, endian issues, etc.)
- Compact (stores the bits; lossless compression possible)
- Fast (no extra conversion)
- Partial read & write (some of them, important for big data)
- Standard & user defined places to put metadata!

Filenames

- It is useful to have some form of metadata in filename
  - File number, date, etc.
- Too easy to overwrite
- Store all metadata in file headers
  - Copy useful subset into file name for convenience
  - If internal metadata and file name disagree, internal wins

Provenance

Metadata goals

- **Basic:** you can read the information about your data from the file (nutrition label for everything on a plot)
- **Goal:** you can recreate the analysis as needed (git hashes, etc.)
- **Advanced:** can recreate full instrument & analysis state (e.g. adds links back into Monitor & Control database)

} **Provenance**

You can recreate your analysis

Provenance examples

- Instrumental settings & environment
  - control knob settings
  - temperatures/voltages/field strengths etc.
  - component versions, identifiers & connectivity
  - timestamps
- code
  - **full** code version information
  - command-line arguments & keywords
  - timestamp of when the code was run
- version information for any code/database/file used as an external input to the analysis
- *full stack: versions of os & external libraries*

History table pattern

- Header(s) for data and instrument information
- History header that each piece of code appends to
  - **full** code version information (version + git hash)
  - command-line arguments & keywords
  - timestamp of when the code was run

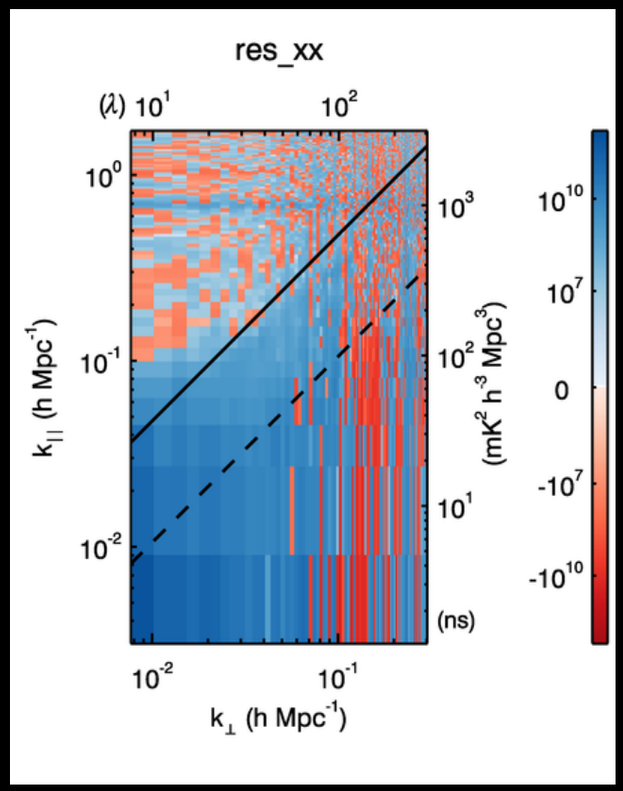
Analysis traceability

- Code that ran on file
- All command-line arguments & settings
- code version
- githash

Data unit tests

2 fhd_core/fhd_struct_init_antenna.pro View

```
@@ -86,7 +86,7 @@ dec_use=dec_arr[valid_i]
 ;NOTE: Eq2Hor REQUIRES Jdate to have the same number of elements as RA and Dec for precession!!
 ;;NOTE: The NEW Eq2Hor REQUIRES Jdate to be a scalar! They created a new bug when they fixed the old one
-Eq2Hor,ra_use,dec_use,Jdate,alt_arr1,az_arr1,lat=obs.lat,lon=obs.lon,alt=obs.alt,precess=1
+Eq2Hor,ra_use,dec_use,Jdate,alt_arr1,az_arr1,lat=obs.lat,lon=obs.lon,alt=obs.alt,precess=1,/nutate
 za_arr=fltarr(psf_image_dim,psf_image_dim)+90. & za_arr[valid_i]=90.-alt_arr1
 az_arr=fltarr(psf_image_dim,psf_image_dim) & az_arr[valid_i]=az_arr1
```



Hazelton

From pygitversion (based on setuptools_scm)

On main for pyuvdata:

```
pyuvdata.__version__
Out[2]: ‘2.1.6.dev5+g406b88eb’
```

On multi_source branch for pyuvdata

```
pyuvdata.__version__
Out[2]: ‘2.1.6.dev88+g71ef0ba4.multi_source'
```

https://github.com/RadioAstronomySoftwareGroup/pygitversion#readme

Test thickets

Make it idiot proof and someone will make a better idiot

Test thickets combine:

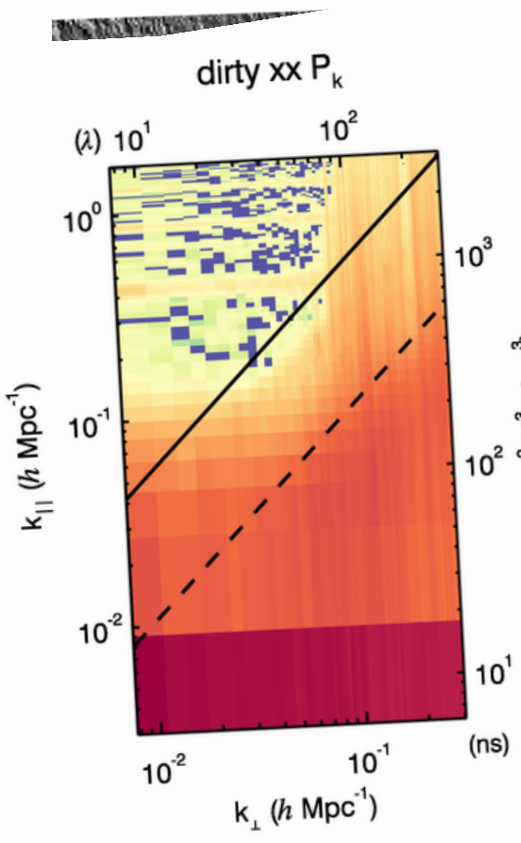
- Worry tests
- Visualization
- Provenance

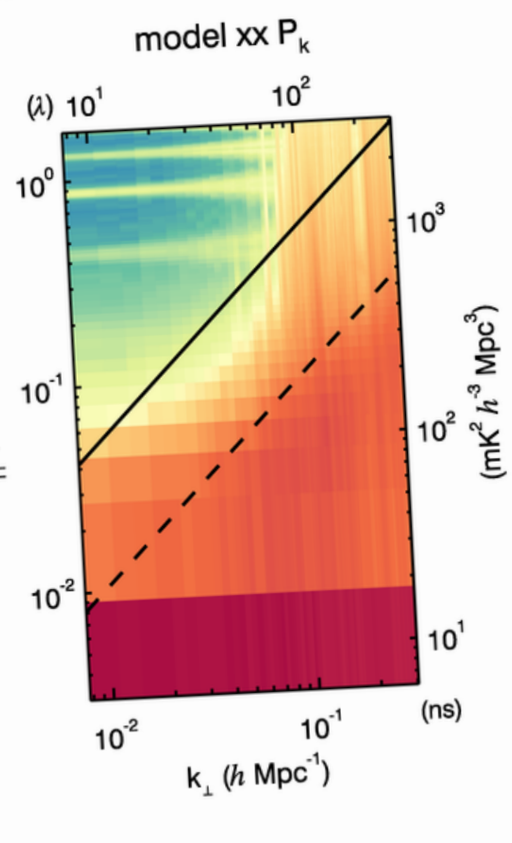
Test thickets

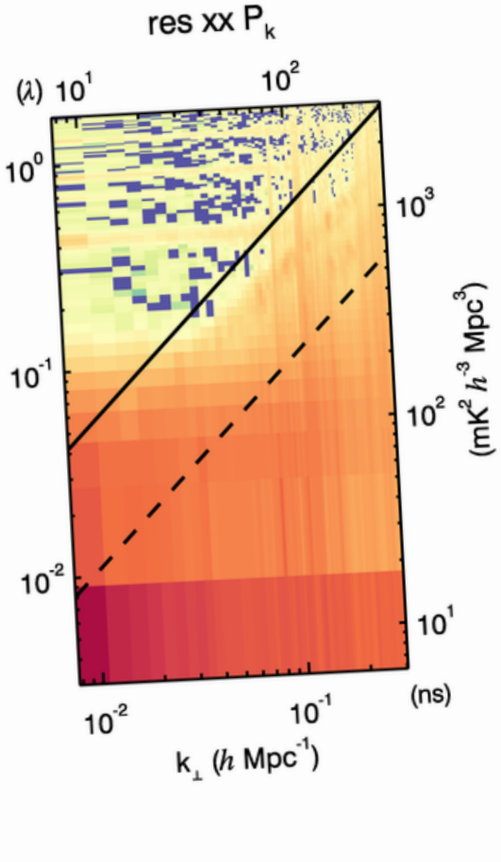
- Capture good worry tests, and run *every* time
- Don’t be too clever, sanity tests are great
- Goal 1: pre-calculate the first tests you would perform if something looks off
- Goal 2: make any major problem obvious—cover possible screw ups

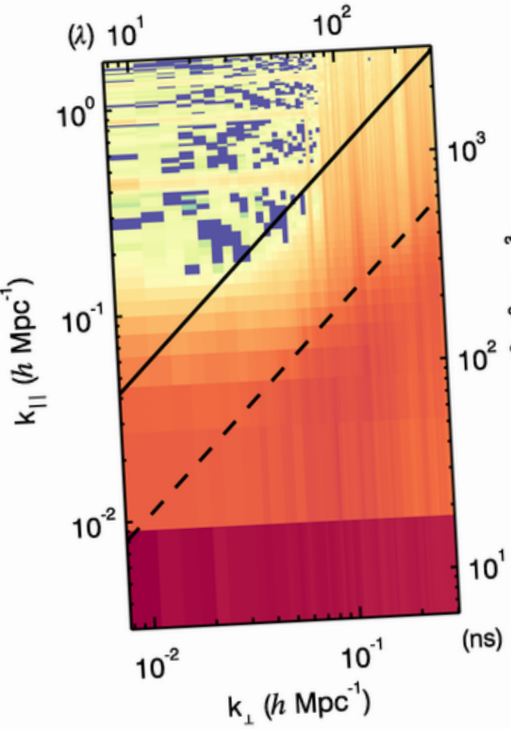
Testing thicket

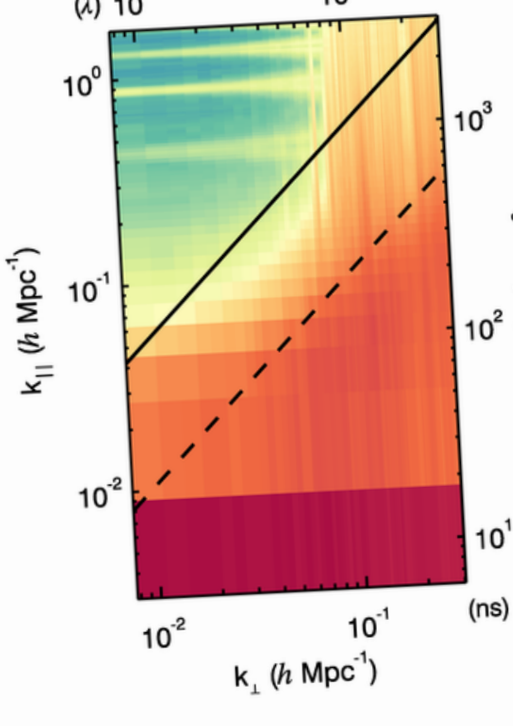
- Make good plots an integral part of your analysis
- Diversity of data views key
- Way of loving large data sets

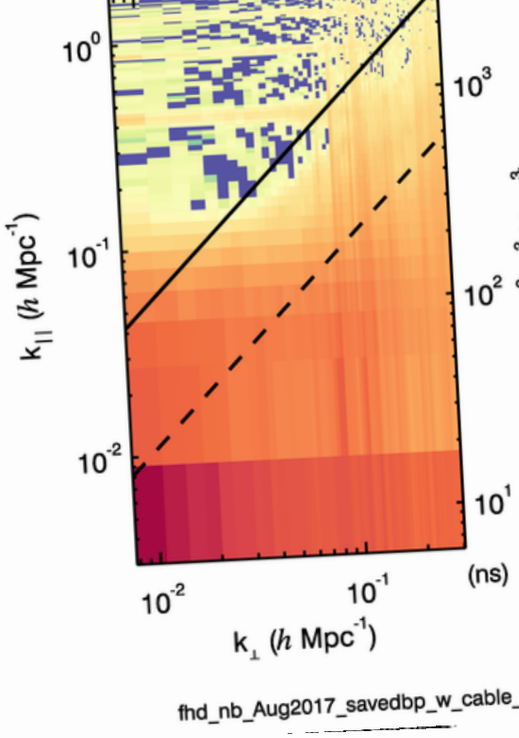

dirty xx P_k
(λ) 10^1
10^2
10^0
10^-1
10^-2
10^3
10^2
10^1
k_|| (h Mpc^-1)
k_⊥ (h Mpc^-1)
(ns)
(mK^2 h^-3 Mpc^3)
10^15
10^13
10^11
10^9
10^7
10^5
10^3
<0


model xx P_k
(λ) 10^1
10^2
10^0
10^-1
10^-2
10^3
10^2
10^1
k_|| (h Mpc^-1)
k_⊥ (h Mpc^-1)
(ns)
(mK^2 h^-3 Mpc^3)
10^15
10^13
10^11
10^9
10^7
10^5
10^3

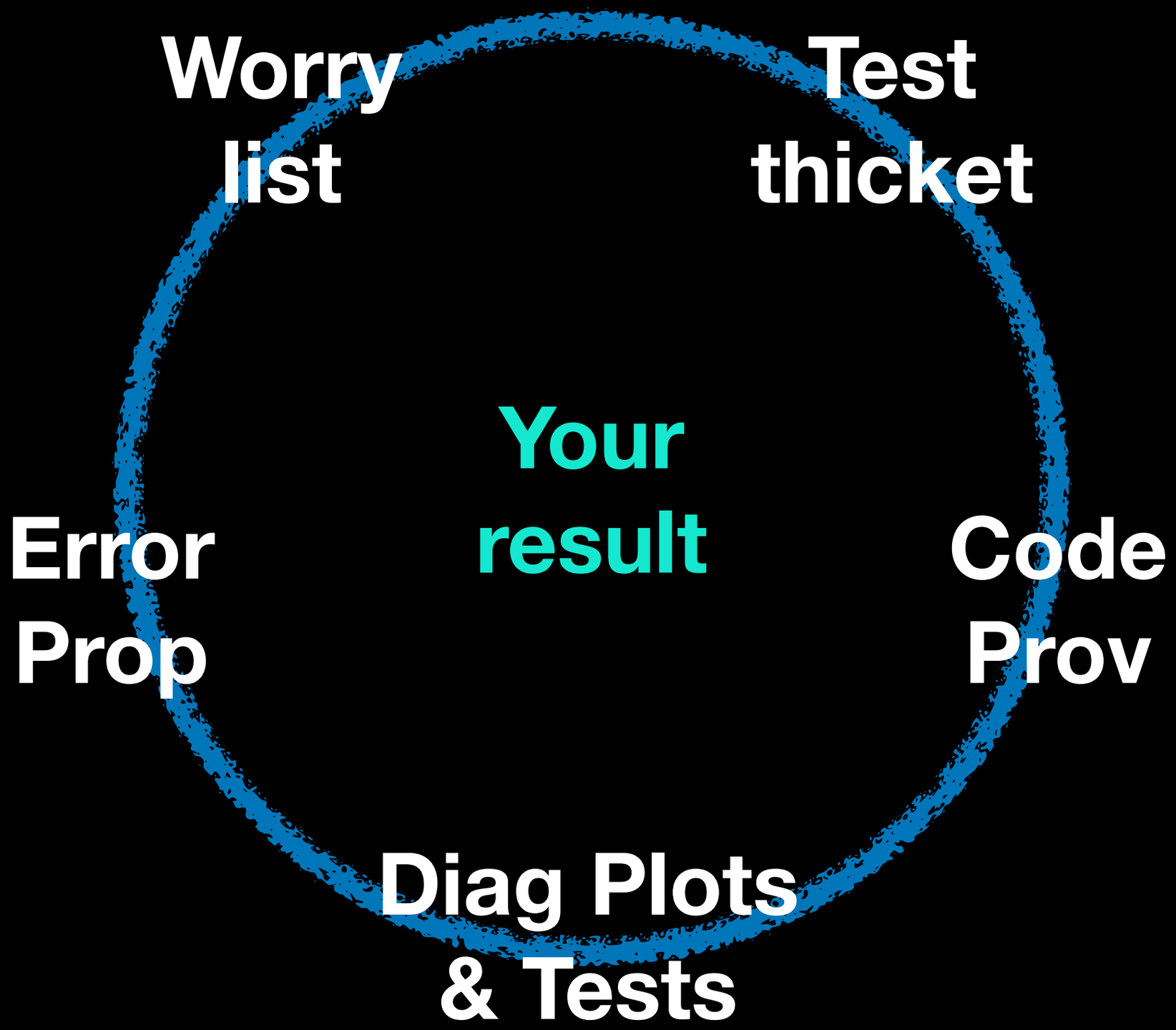

res xx P_k
(λ) 10^1
10^2
10^0
10^-1
10^-2
10^3
10^2
10^1
k_|| (h Mpc^-1)
k_⊥ (h Mpc^-1)
(ns)
(mK^2 h^-3 Mpc^3)
10^15
10^13
10^11
10^9
10^7
10^5
10^3
<0


dirty yy P_k
(λ) 10^1
10^2
10^0
10^-1
10^-2
10^3
10^2
10^1
k_|| (h Mpc^-1)
k_⊥ (h Mpc^-1)
(ns)
(mK^2 h^-3 Mpc^3)
10^15
10^13
10^11
10^9
10^7
10^5
10^3
<0


model yy P_k
(λ) 10^1
10^2
10^0
10^-1
10^-2
10^3
10^2
10^1
k_|| (h Mpc^-1)
k_⊥ (h Mpc^-1)
(ns)
(mK^2 h^-3 Mpc^3)
10^15
10^13
10^11
10^9
10^7
10^5
10^3


res yy P_k
(λ) 10^1
10^2
10^0
10^-1
10^-2
10^3
10^2
10^1
k_|| (h Mpc^-1)
k_⊥ (h Mpc^-1)
(ns)
(mK^2 h^-3 Mpc^3)
10^15
10^13
10^11
10^9
10^7
10^5
10^3
<0

Protecting a result



Final Presentation

- 20 min (5 min intro; 10 min analysis you are doing; 5 min questions)

- Email me if: early or late preference, or don't want to present