Class 11: Metadata, Provenance & Test Thickets

Bryna Hazelton
Miguel F. Morales

Metadata

All the information about your data

- When it was taken
- What instrument took it/How it was taken
- Environmental data/State of the instrument (telemetry)
- 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 or jackknife

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, all information needed for a data jackknife)

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

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

Generating and capturing metadata

- Automated when possible
  - more consistent, automate and forget
  - may need intermediate storage, e.g. database
- If you can’t automate, build a system that makes it easy to remember to capture and record it
  - file formats that are both human editable and machine readable, e.g. yaml
  - Integrate it into data files as early in your pipeline as possible

Storing 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)
  - HDF5, FITS, AVRO…
- Some common file formats do not!
  - CSV, tab delimited text files

Use a standard binary file format (HDF5, FITS, 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!
- Use existing standards in your field as much as possible
- Be as consistent as possible with field names

Filenames

- It is useful to have some 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

Standard identifiers

- Standard identifiers can help with linking metadata stored in other locations (if impractical to store in the data file)
    - other files or databases (e.g. monitor and control)
        - e.g. the GPS second that the data were taken
- need to be unique for a dataset
- better to be meaningful rather than arbitrary

Provenance

Metadata goals

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

- **Goal:** you can recreate the analysis as needed (git hashes, etc. — supports analysis jackknife)

- **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 (telemetry)
  - 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

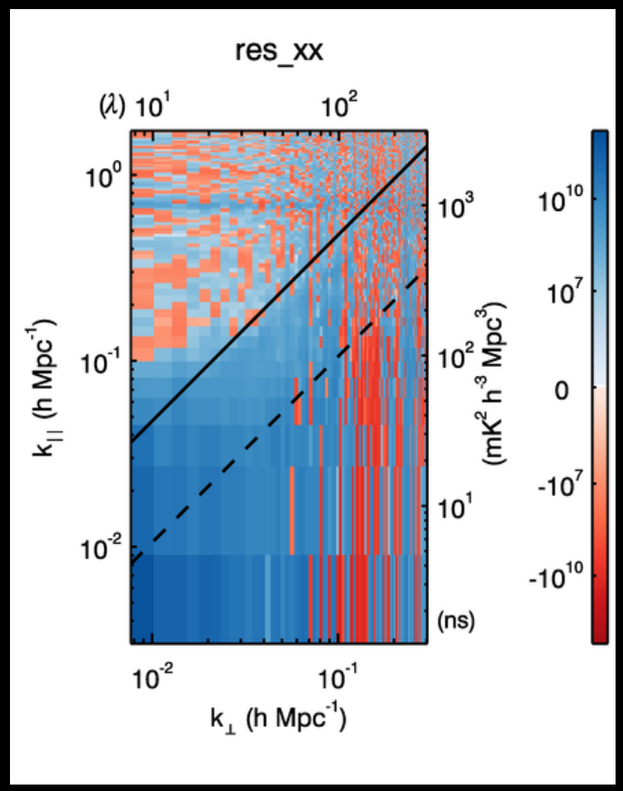
Analysis traceability

- Code that ran to produce file
- All command-line arguments & settings
- code version
- git hash (unique identifier of a commit)

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

In python, use setuptools_scm for versioning

- Version string: imost recent git tag + number of commits since that tag + the git hash (uniquely identifies a commit).

- Setup is a little fiddly, in particular it's helpful to set it up so that the `__version__` attribute of a package contains the setuptools_scm derived string.

- In some of our projects we also capture the branch name in the version

    - This requires a little extra code, see https://github.com/RadioAstronomySoftwareGroup/pygitversion

In other languages, call git directly

- origin (e.g. url to the repo on github):
  - `git config --get remote.origin.url <path_to_local_repo>`
- branch
  - `git rev-parse --abbrev-ref HEAD <path_to_local_repo>`
- description (a string with latest tag + number of commits since tag + short hash + indication of local uncommitted changes)
  - `git describe --dirty --tag –always <path_to_local_repo>`
- full hash (not required if you get the description)
  - `git rev-parse HEAD <path_to_local_repo>`

full info: origin + branch + description (tag, # commits since tag, hash)

Test thickets

Make it idiot proof and someone will make a better idiot

Test thickets combine:

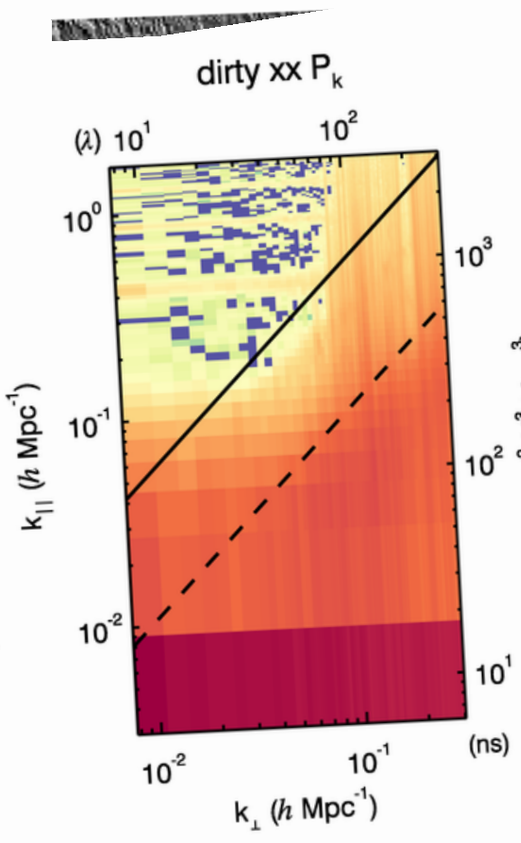
- Worry tests
- Visualization
- Provenance

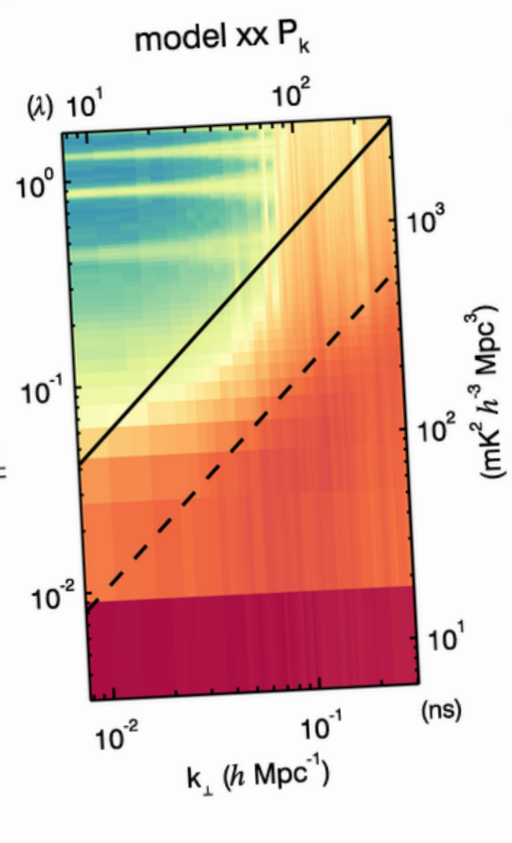
Test thickets

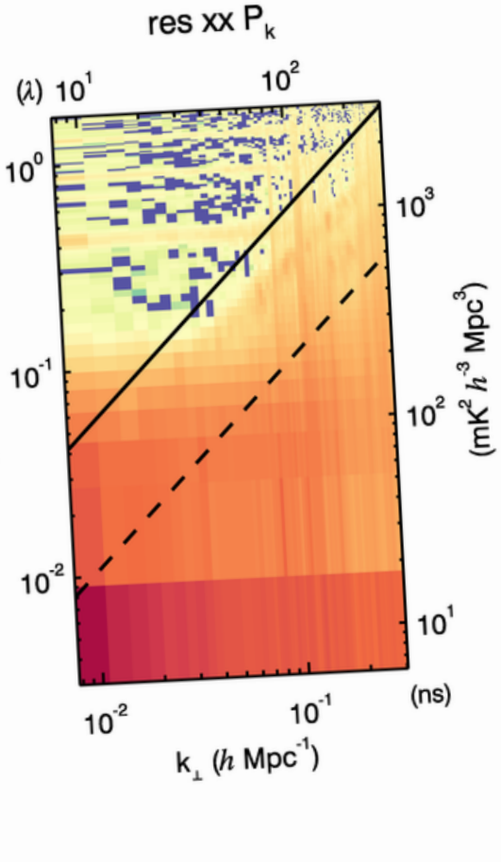
- Capture good worry tests, and run *every* time (automated!)
    - plots/tests that have caught problems before are great: prevent similar mistakes in the future
- Don’t be too clever, sanity tests are great
- Goal 1: pre-calculate the first tests you would perform if something looks off
- Goal 2: make any major problem obvious—cover possible screw ups

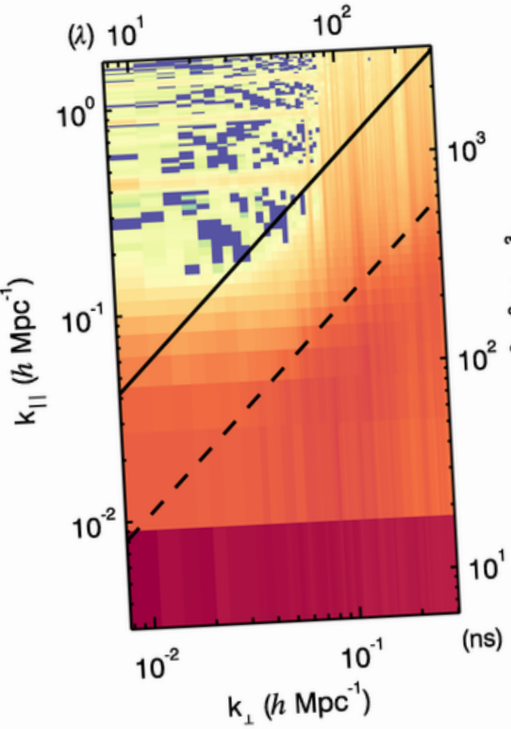
Testing thicket

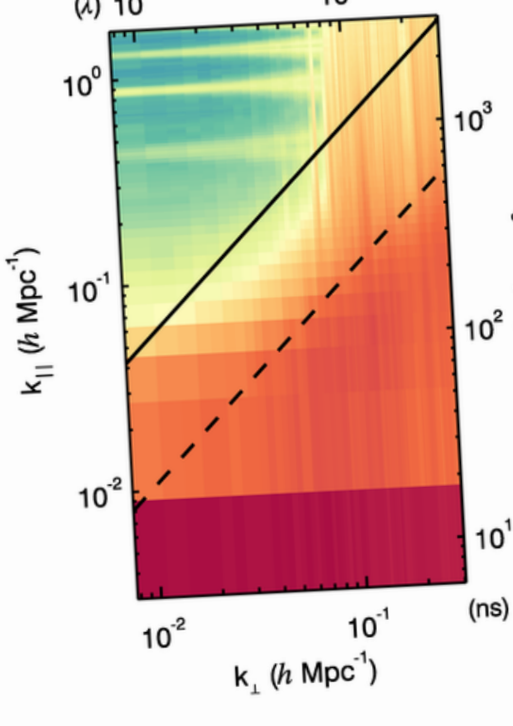
- Make good plots an integral part of your analysis
- Diversity of data views key
- Way of loving large data sets

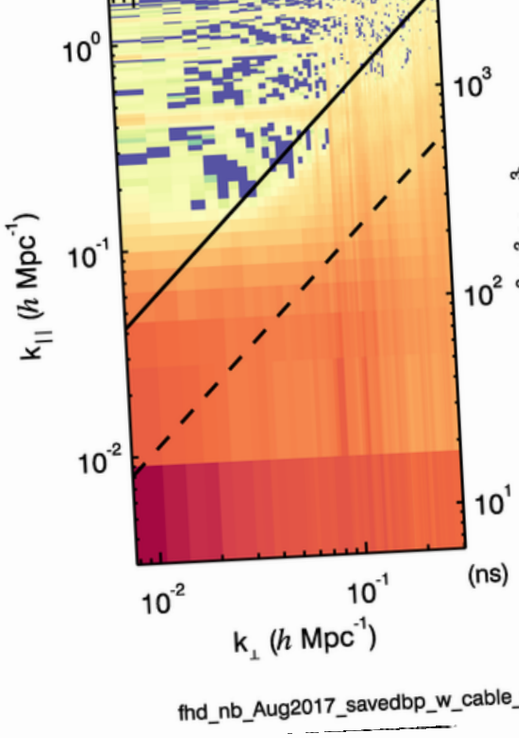

dirty xx P_k
(λ) 10^1
10^2
10^0
10^-1
10^-2
10^3
10^2
10^1
k_|| (h Mpc^-1)
k_⊥ (h Mpc^-1)
(ns)
(mK^2 h^-3 Mpc^3)
10^15
10^13
10^11
10^9
10^7
10^5
10^3
<0


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

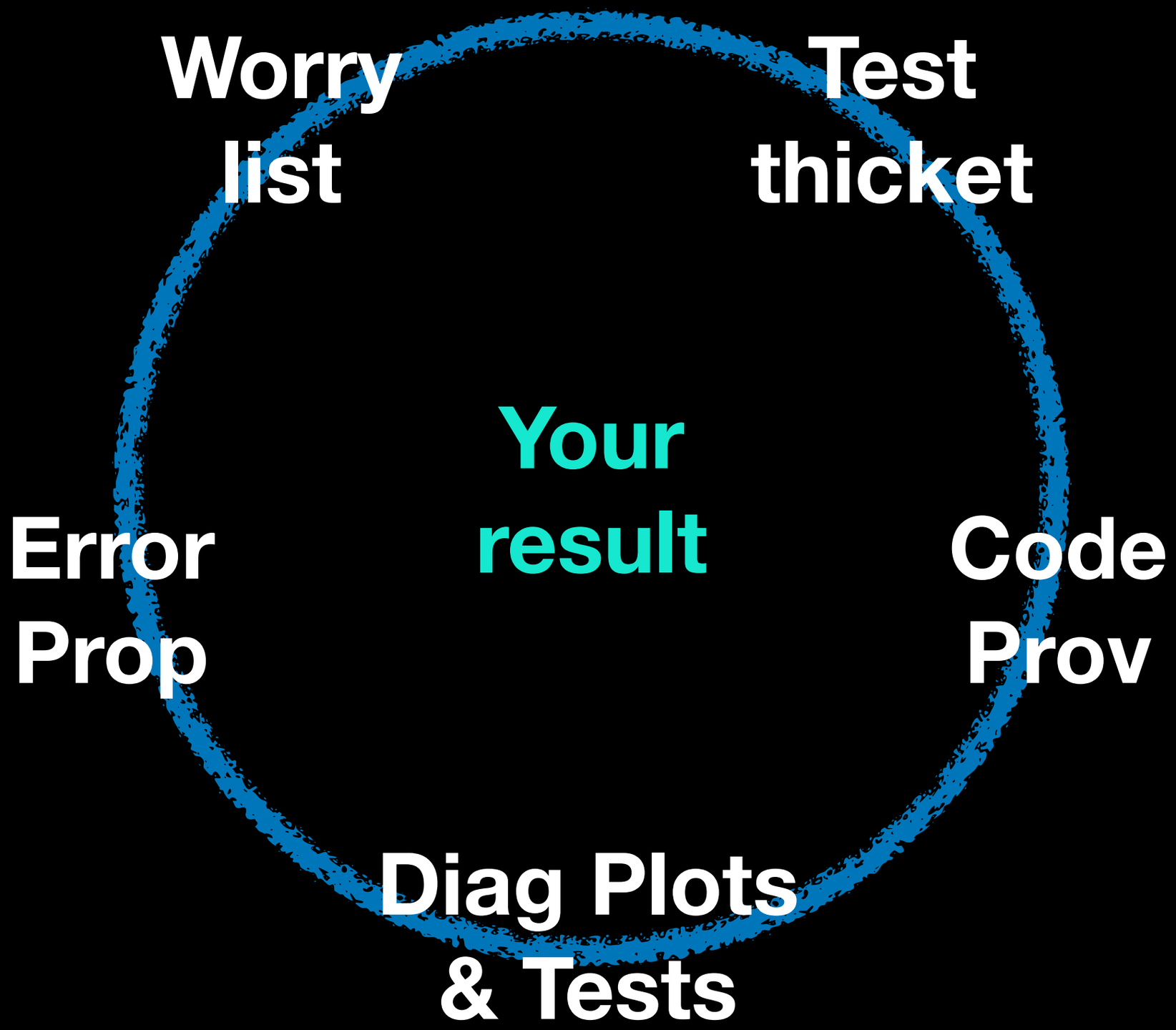

res xx P_k
(λ) 10^1
10^2
10^0
10^-1
10^-2
10^3
10^2
10^1
k_|| (h Mpc^-1)
k_⊥ (h Mpc^-1)
(ns)
(mK^2 h^-3 Mpc^3)
10^15
10^13
10^11
10^9
10^7
10^5
10^3
<0


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