Errors and Bootstrap Resampling

Statistics and Data Science

Spring 2025

Goals for today: you should be able to…

- Explain the difference between random and systematic errors
- **lecture 13/14 notebook**:
  - Work with Pandas dataframes
  - Obtain bootstrap samples in Python

Review: Random and Systematic Errors

- We found for the standard deviation of the mean:

$$\sigma_m^2 = \frac{\sigma^2}{n} + n^{-2}\Sigma_{i\neq j}\ \mathbb{E}(x_i - \mu)(x_j - \mu) = \frac{\sigma^2}{n} + n^{-2}\ \Sigma_{i\neq j}\ \mathrm{cov}[x_i, x_j]$$

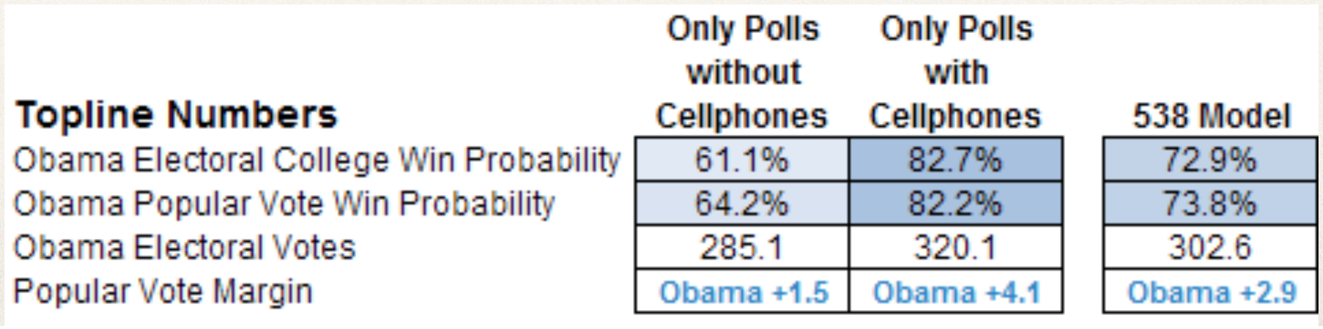
- The standard error (or standard deviation of the mean) gets better as n$^{-1/2}$ as a consequence of the central limit theorem. We call such errors ***random errors***: they obey typical statistical rules of thumb.
- Other sources of uncertainty may not get better by getting more data. These are often calibration uncertainties or physical effects whose magnitude we can only guess at and will be ***covariant*** between different data points. We refer to these as ***systematic errors***.

Random and Systematic errors

- If we ignore the possibility of clock errors, we'd calculate the uncertainty in the time delay to be much smaller than its true value (~4 ns from the 20 events).
- The standard error (or standard deviation of the mean) gets better as $n^{-1/2}$ as a consequence of the central limit theorem. We call such errors ***random errors***: they obey typical statistical rules of thumb.
- Other sources of uncertainty may not get better by getting more data. These are often calibration uncertainties or physical effects whose magnitude we can only guess at. We refer to these as ***systematic errors***.
- **Be wary of papers that do not consider the possibility of (or estimate the magnitude of) systematic errors!**

Examples of systematic errors

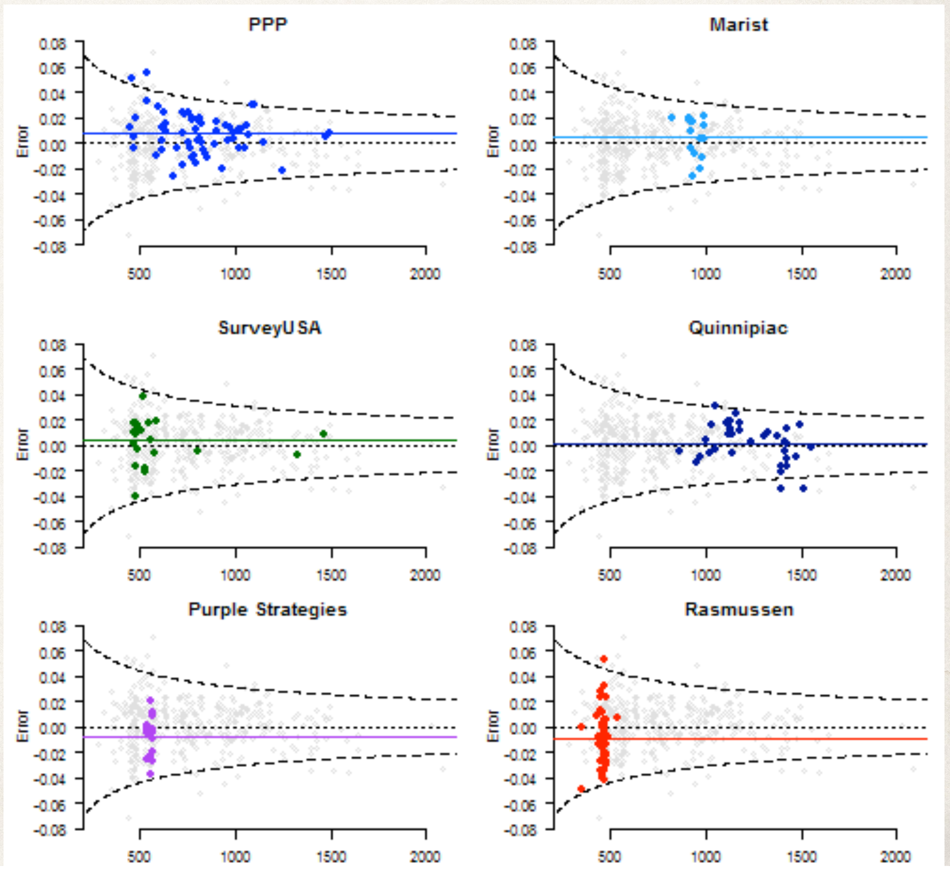
- Many polls used to only dial land-line telephone numbers, not cellphones; so people who only have cell phones wouldn't get counted.
- This wouldn't make a difference if they favored one candidate at the same rate as everyone else; polls would still be a fair sample (think about this in terms of covariance between phone type and vote preference).
- Instead, in 2012 polls that included cell phones favored Obama by 2-3 points more than polls that didn't (see: http://fivethirtyeight.blogs.nytimes.com/2012/09/19/obamas-lead-looks-stronger-in-polls-that-include-cellphones/)
- Even with a huge, land-line only sample, this effect would never go away



Topline Numbers	Only Polls without Cellphones	Only Polls with Cellphones	538 Model
Obama Electoral College Win Probability	61.1%	82.7%	72.9%
Obama Popular Vote Win Probability	64.2%	82.2%	73.8%
Obama Electoral Votes	285.1	320.1	302.6
Popular Vote Margin	Obama +1.5	Obama +4.1	Obama +2.9

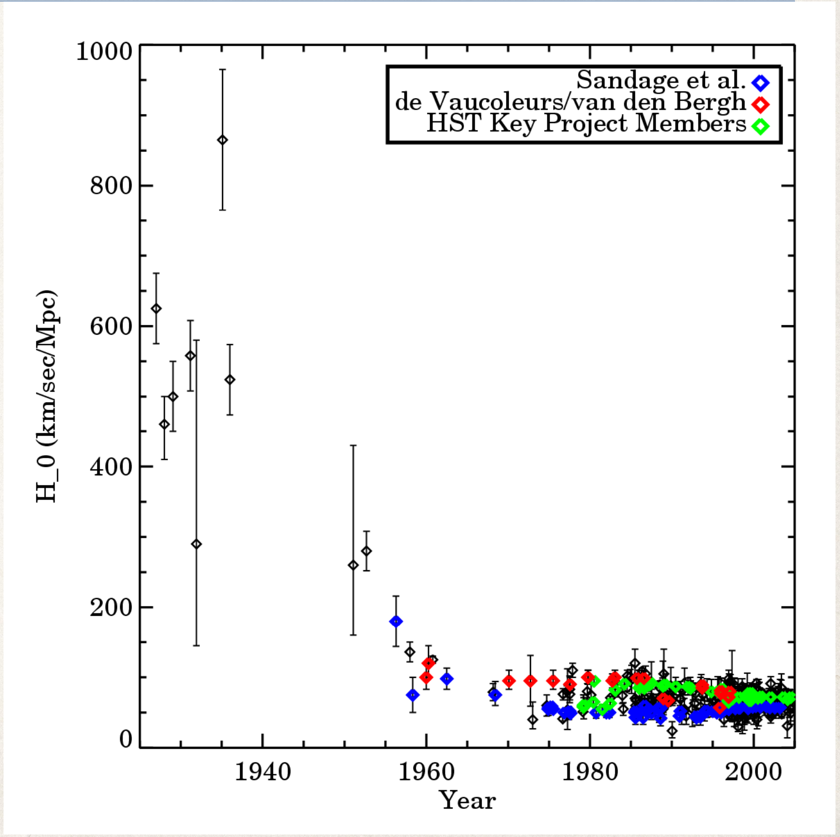
Examples of systematic errors

- Different pollsters' results tend to differ from the mean in different ways
- Generally less than random error in a single poll (dashed lines: expected 95% range vs N) but detectable in the mean (colored line) in some cases
- see http://votamatic.org/looking-for-house-effects/



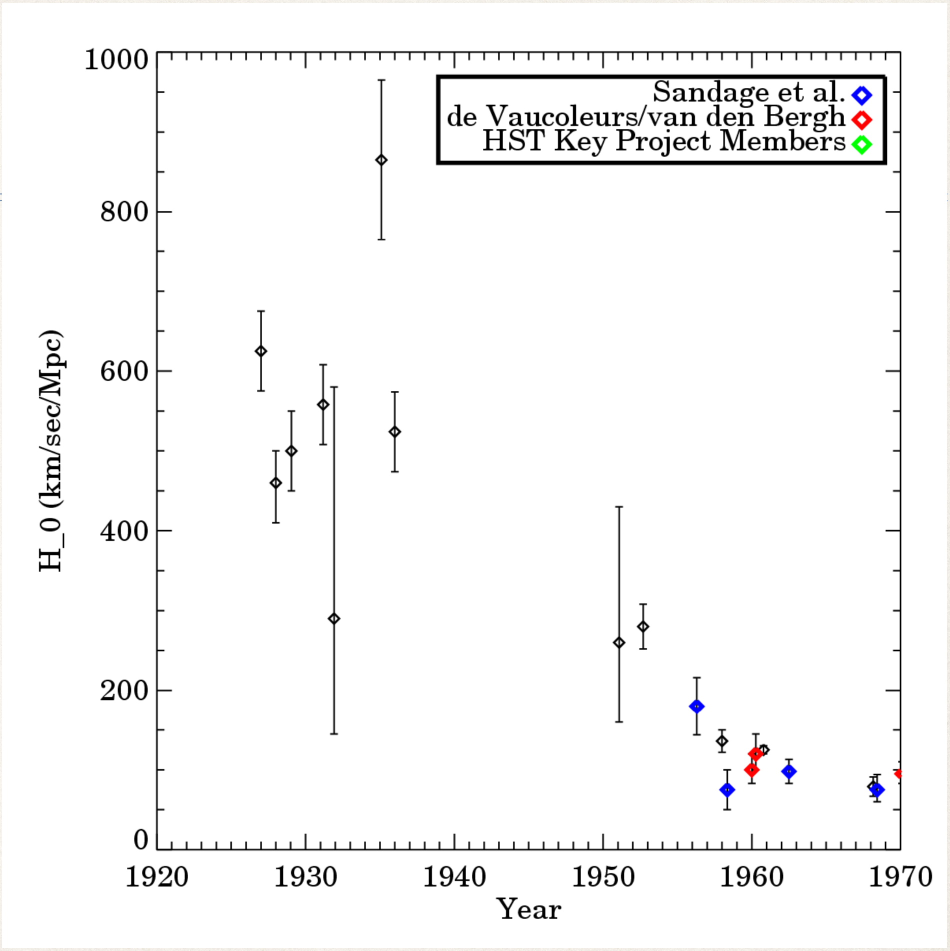
Random and Systematic Errors in Science: The Hubble Constant

- Hubble's Constant H_0 is the ratio of the apparent recession velocity to the distance for distant galaxies (so $v=H_0d$); it measures the expansion rate of the Universe.
  - The first estimates of H_0 were too high by 8-10x. This was because of a systematic error: Hubble was looking at "Population I", high-metal content Cepheids, which are much more luminous than the "Population II" Cepheids with 'known' distances he used to calibrate
    - So he systematically underestimated all distances, and overestimated H_0.



Changes in H_0 measurements, 1920-1970

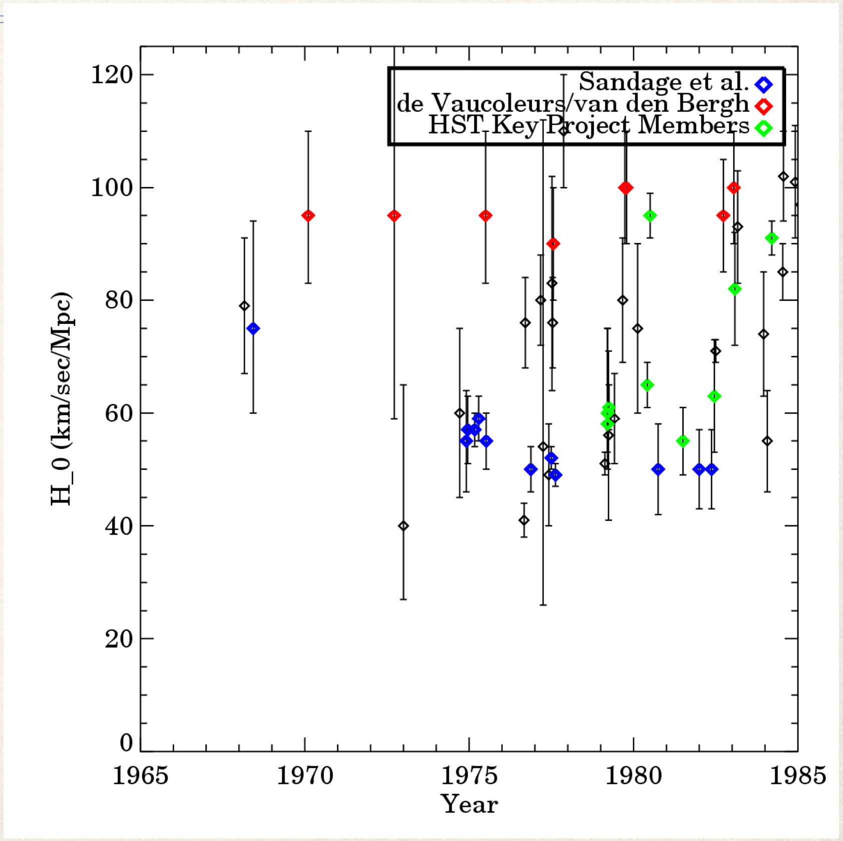
- For many years, the field was dominated by Hubble and his student, Alan Sandage; estimated values of H0 steadily decreased



Data collated by John Huchra

Changes in H_0 measurements, 1965-1985

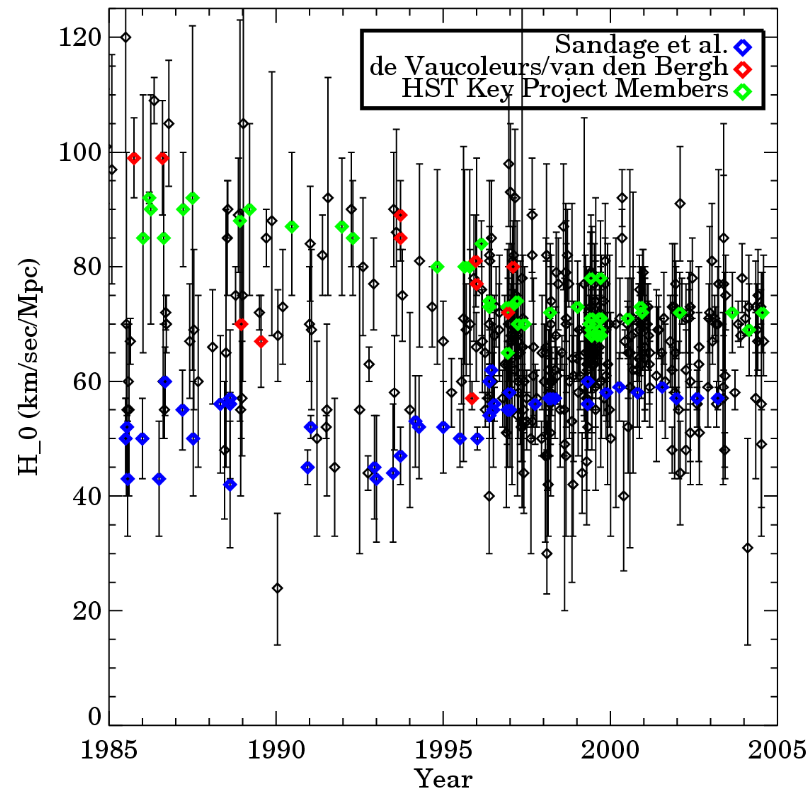
- By the 1980's, 2 camps had developed, repeatedly measuring similar values from many different techniques - but differing amongst each other.



Changes in H_0 measurements 1985-2005

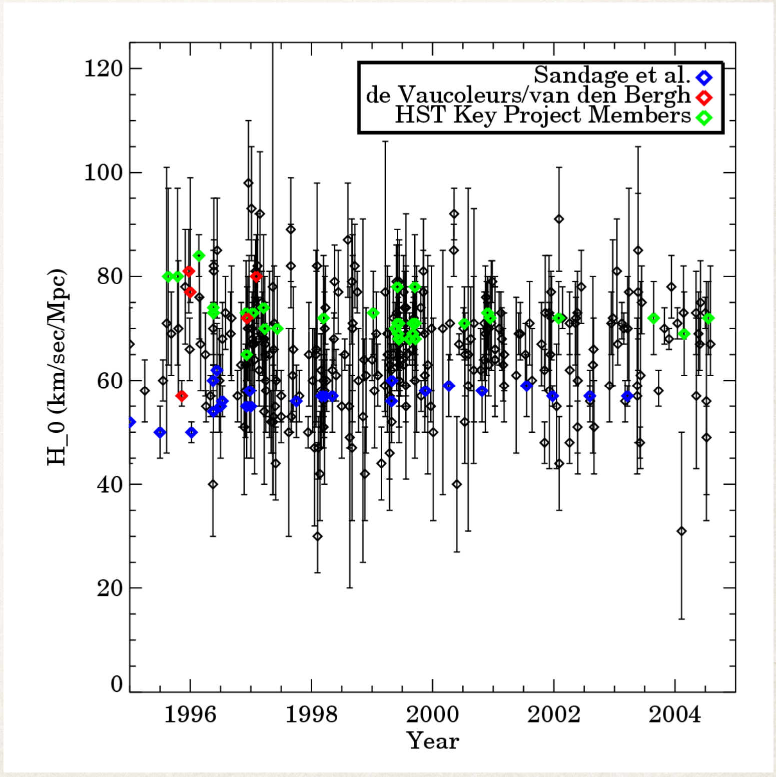
- One of the main purposes for the Hubble Space Telescope was to measure H_0 using Cepheid variable stars as "standard candles": sources of known luminosity from which one could derive distances.

- Two teams - the "Key Project" led by Wendy Freedman & another led by Sandage - intensively used HST to measure the distances to galaxies, which then could be used to calibrate distance indicators for galaxies even further away.



Changes in H_0 measurements, 1995-2005

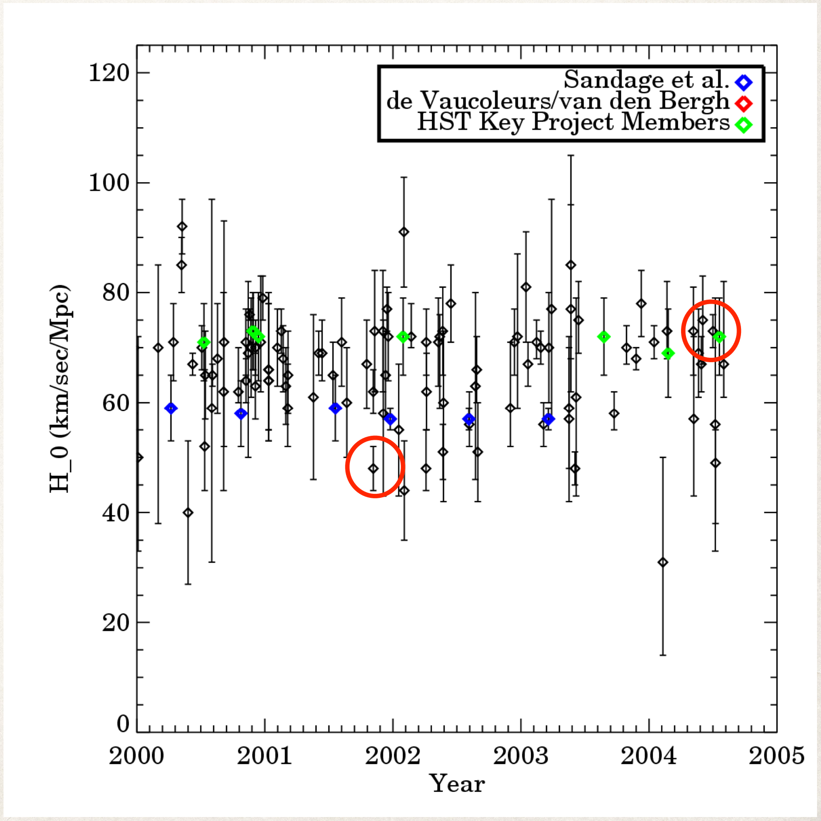
- With more data, results converged some - the camps became 58 vs. 72 km/sec/Mpc, instead of 50 vs. 100.



Changes in H_0 measurements, 2000-2005

- Even in more recent years, many-sigma differences persist - sometimes between different methods or different groups. Clearly, some of these results are simply incorrect, or ignoring large systematics in their errors.

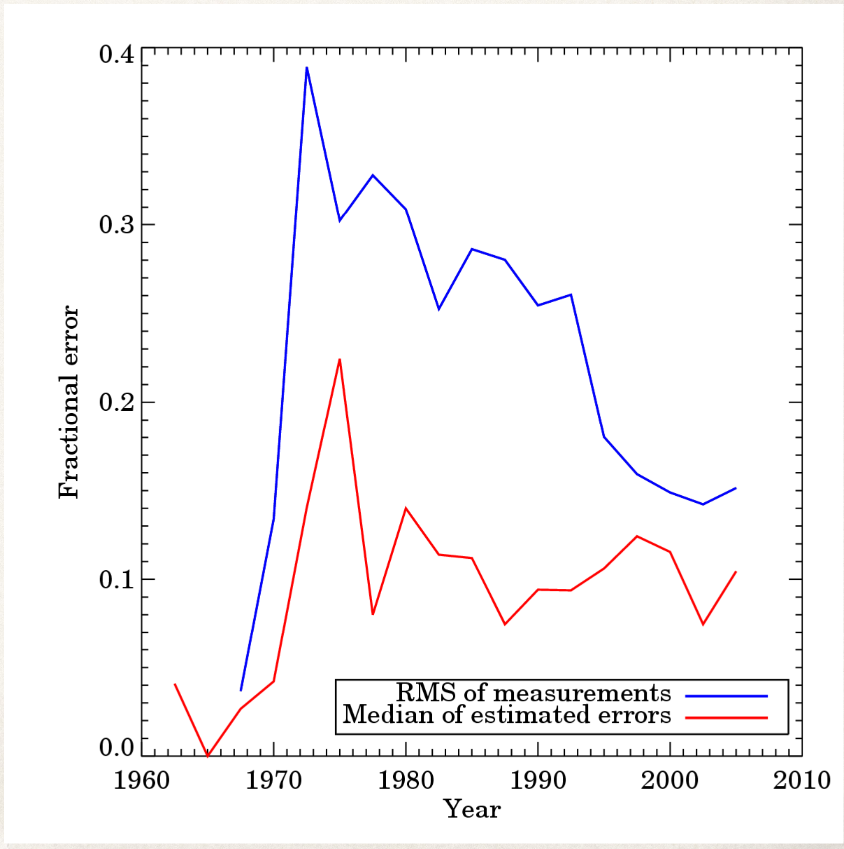
Don't trust every paper you read!



Trends over time: errors

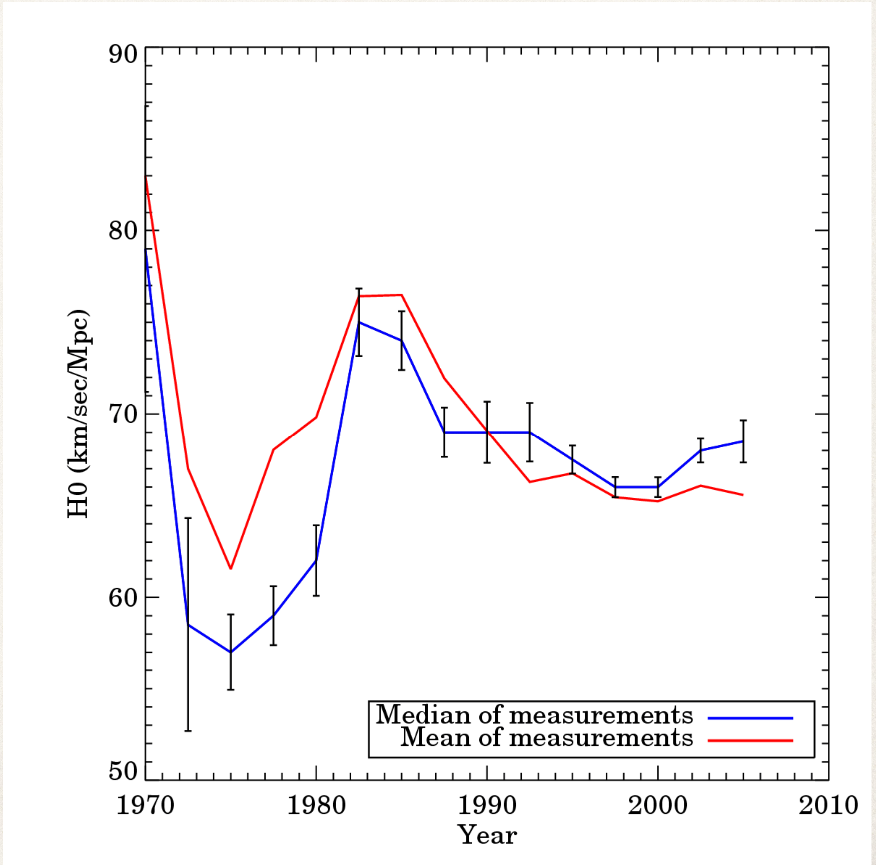
- The goal of the Key Project was to measure H_0 to 10%.
- The typical paper has claimed that accuracy going back to ~1980 (**red curve**)!
- However, until recently, the actual scatter among measurements was 2-3x this high, and it is still ~1.5x higher in modern papers (**blue curve**).

Most people underestimate their errors!



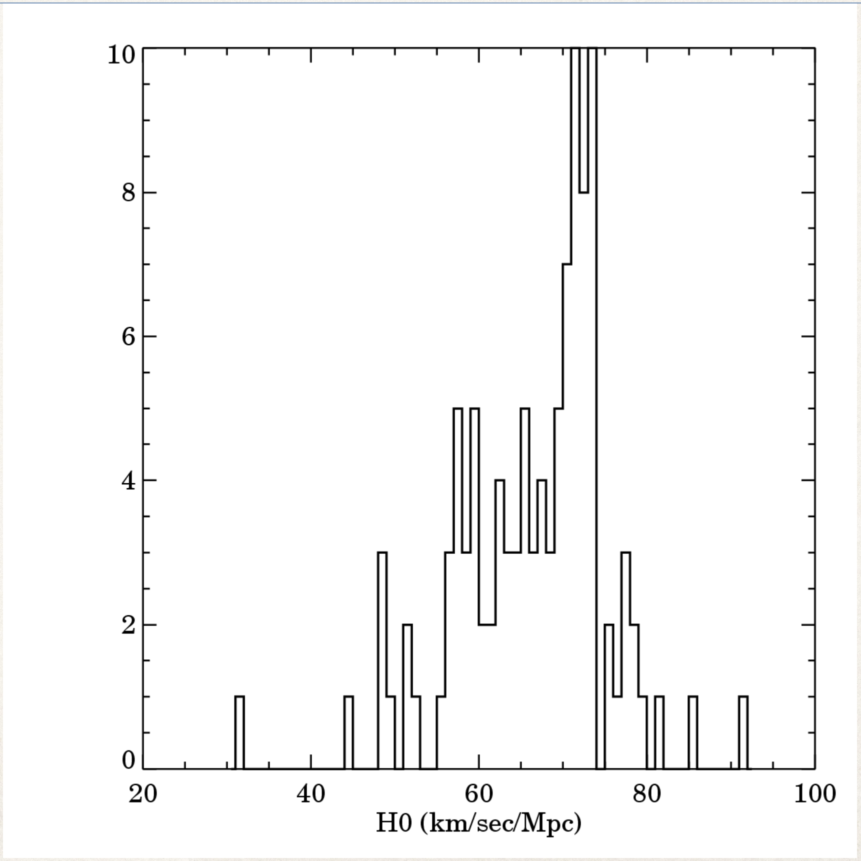
Trends over time: H_0

- If we take all the H_0 measurements in a given period and blindly average them, we can see that the typical H_0 value hasn't changed much since the late 1980's.
- However, this plot has one warning sign: the mean and median values of H_0 differ by more than 1 σ. This will generally only happen in substantially non-Gaussian distributions.

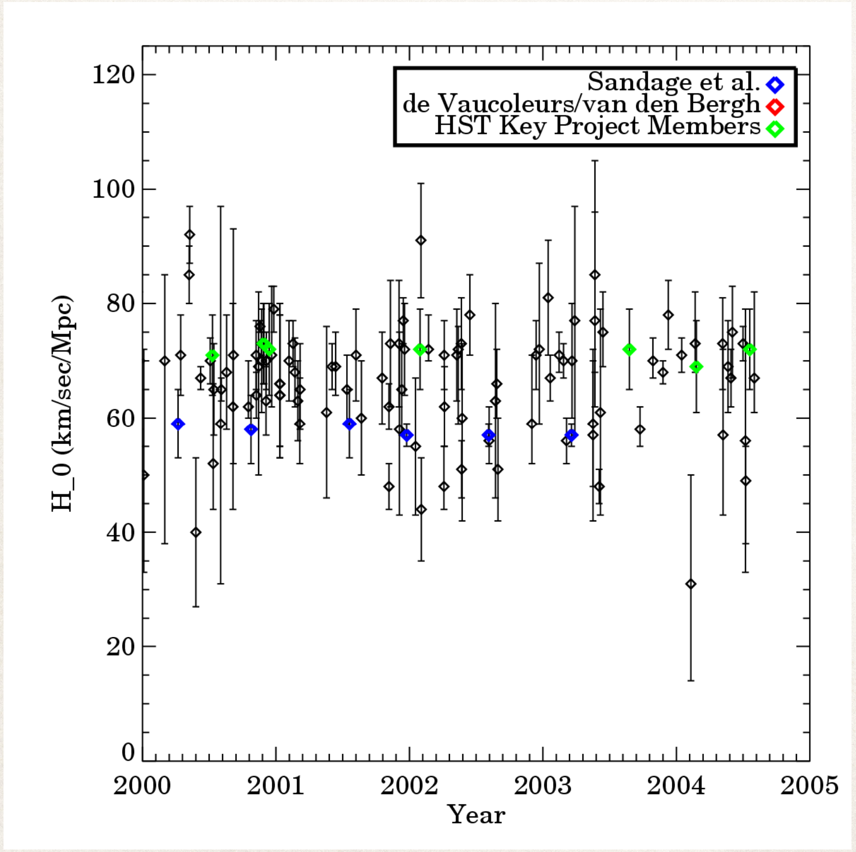


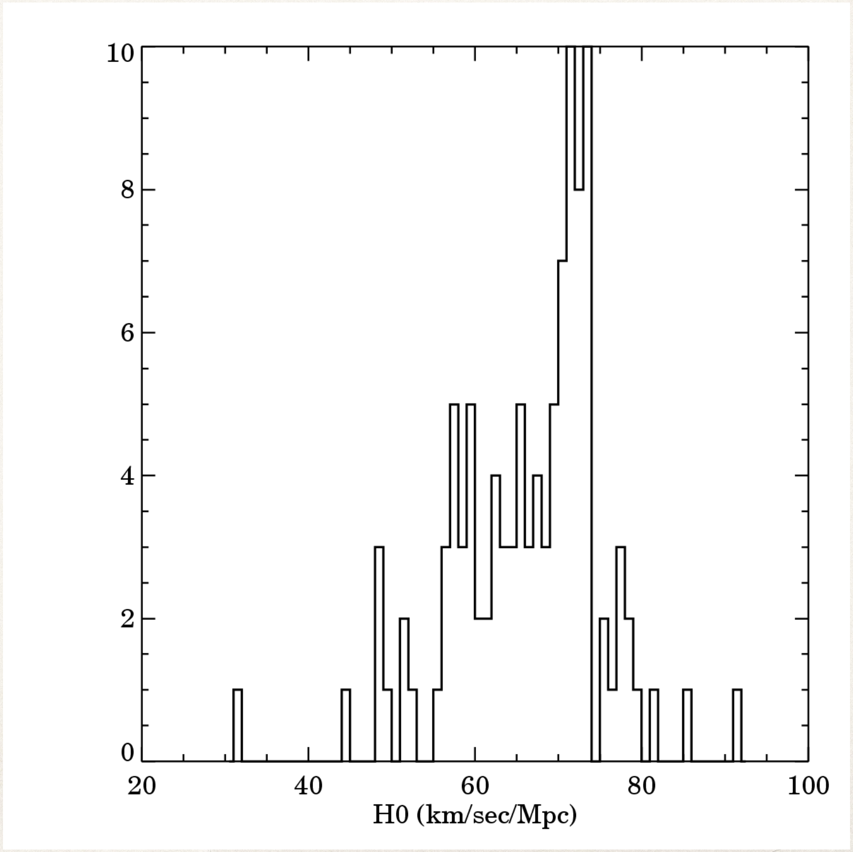
Distribution of H_0 measurements

- The distribution of H_0 measurements since the final Key Project results shows this well.
  - Looks nothing like a single Gaussian distribution!
- Given that typical quoted errors are still ~10%, the sharp spike around the Key Project value is worrying - have people been self-censoring more deviant results, searching for mistakes till they match the Key Project, or what?



Evolution of H_0, 2000-2005





How big do we think systematics are?

- A good practice is to produce a table explicitly listing all possible systematic errors and their contributions: an 'error budget'
- Some people add the magnitudes of all systematic errors directly, yielding a worst-case estimate: 24.5% in this case.
- More common is to add errors in quadrature (so $\sigma_{sys}^2=\Sigma\sigma_i^2$), yielding 10.2% for the Key Project

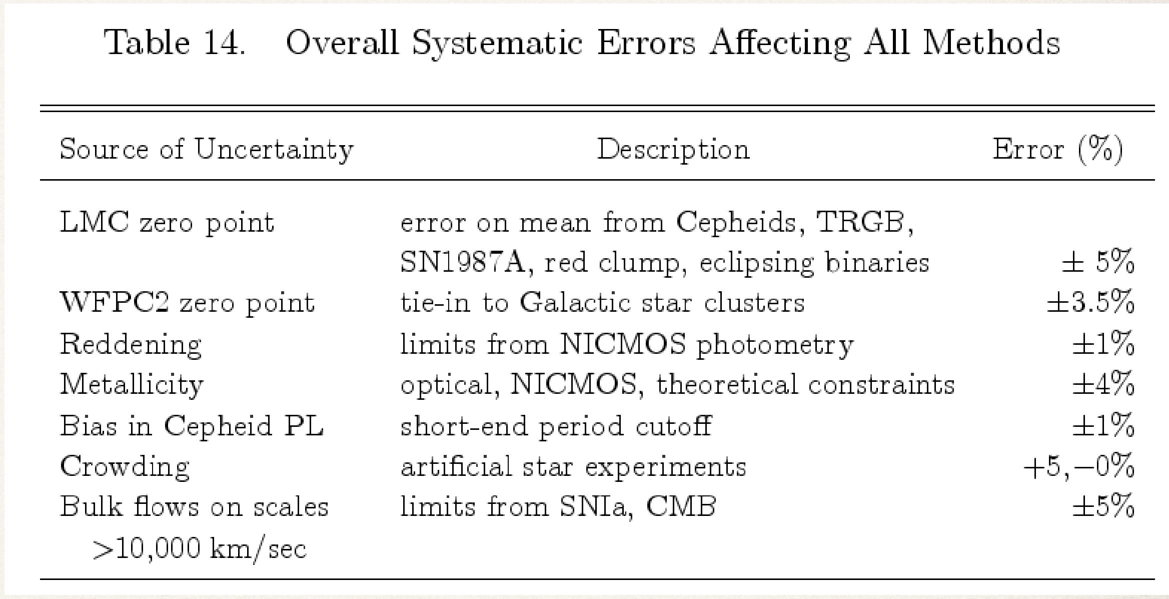


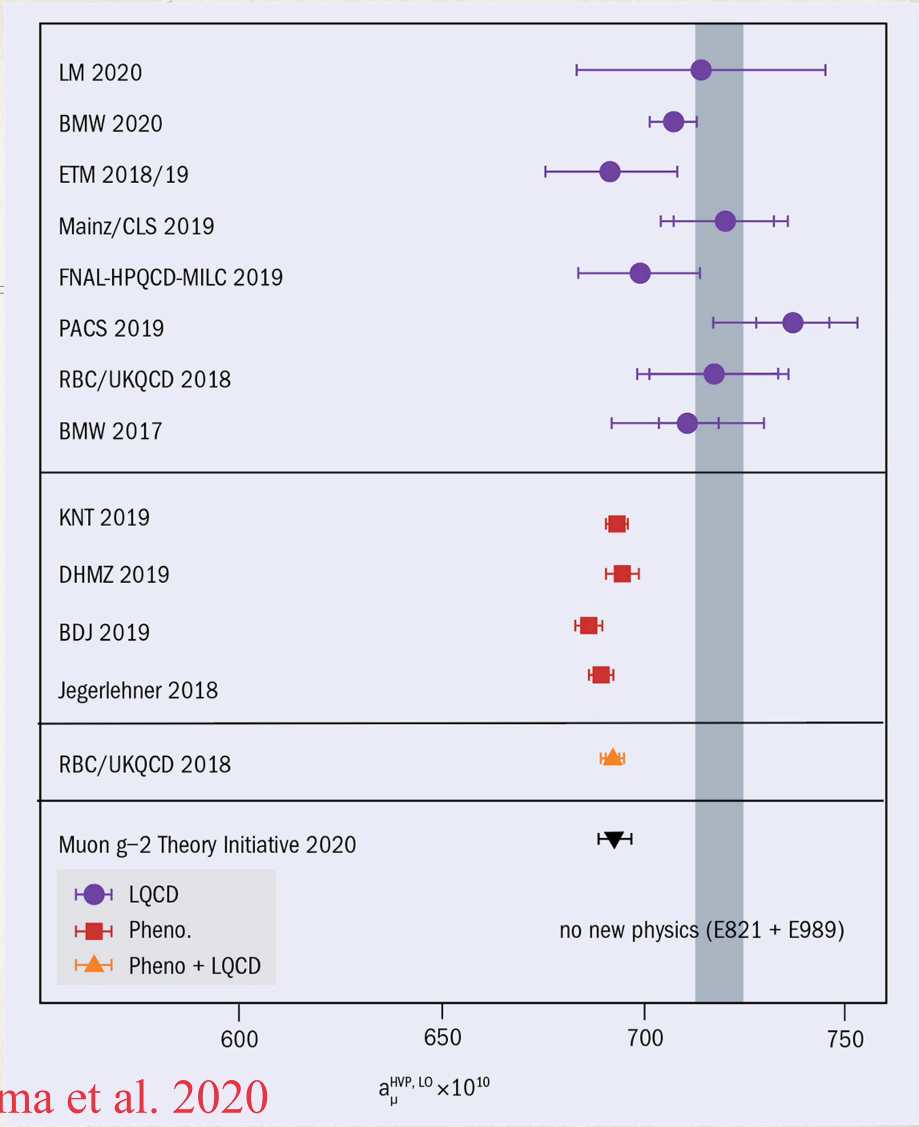
Table 14. Overall Systematic Errors Affecting All Methods

Source of Uncertainty	Description	Error (%)
LMC zero point	error on mean from Cepheids, TRGB, SN1987A, red clump, eclipsing binaries	± 5%
WFPC2 zero point	tie-in to Galactic star clusters	±3.5%
Reddening	limits from NICMOS photometry	±1%
Metallicity	optical, NICMOS, theoretical constraints	±4%
Bias in Cepheid PL	short-end period cutoff	±1%
Crowding	artificial star experiments	+5,−0%
Bulk flows on scales >10,000 km/sec	limits from SNIa, CMB	±5%

Freedman et al. 2001

These issues are not unique to astrophysics

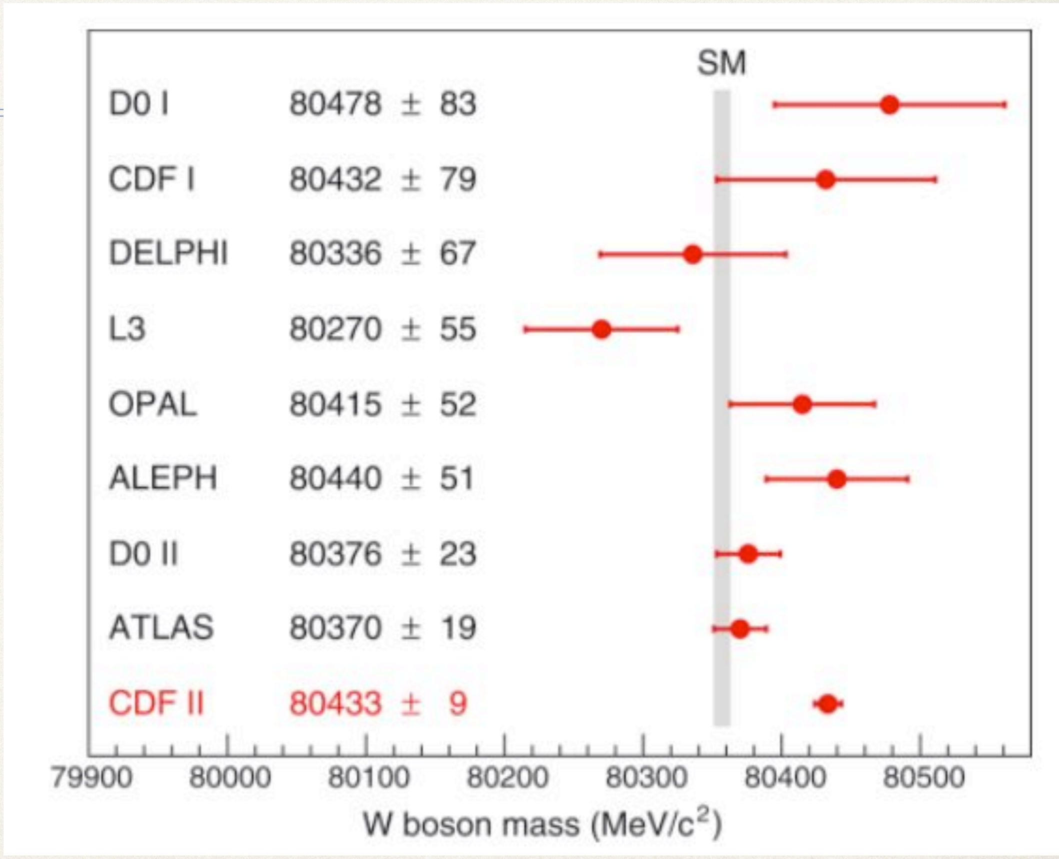
- Plotted are a variety of recent estimates of hadronic vacuum polarization contributions to the muon magnetic moment
- Differences are inconsistent with estimated errors



Aoyama et al. 2020

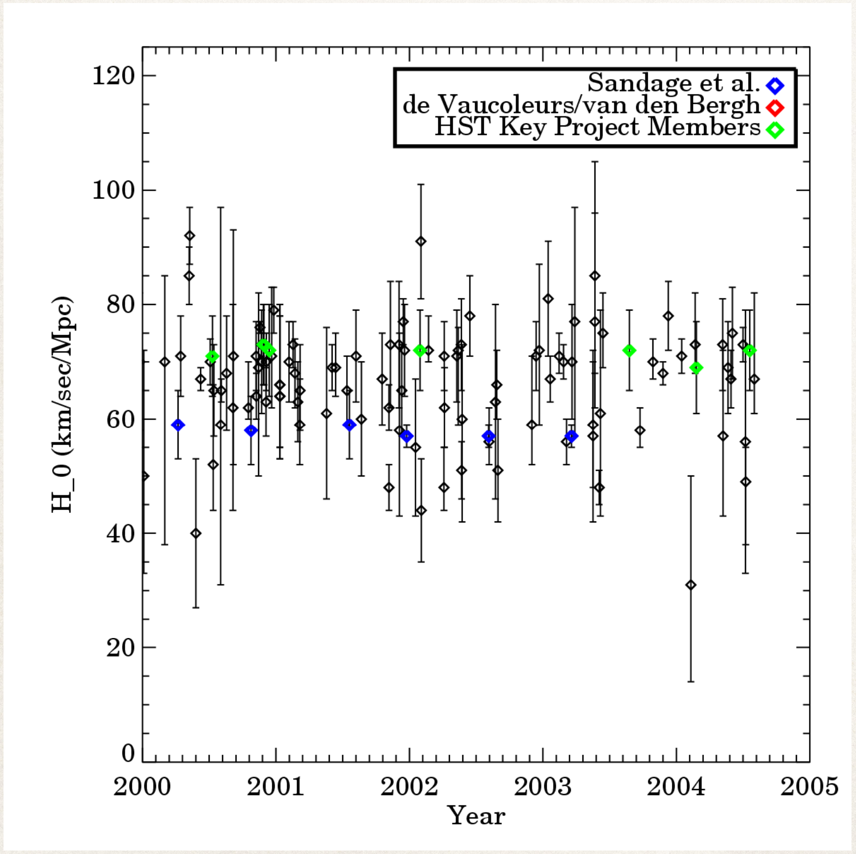
These issues are not unique to astrophysics

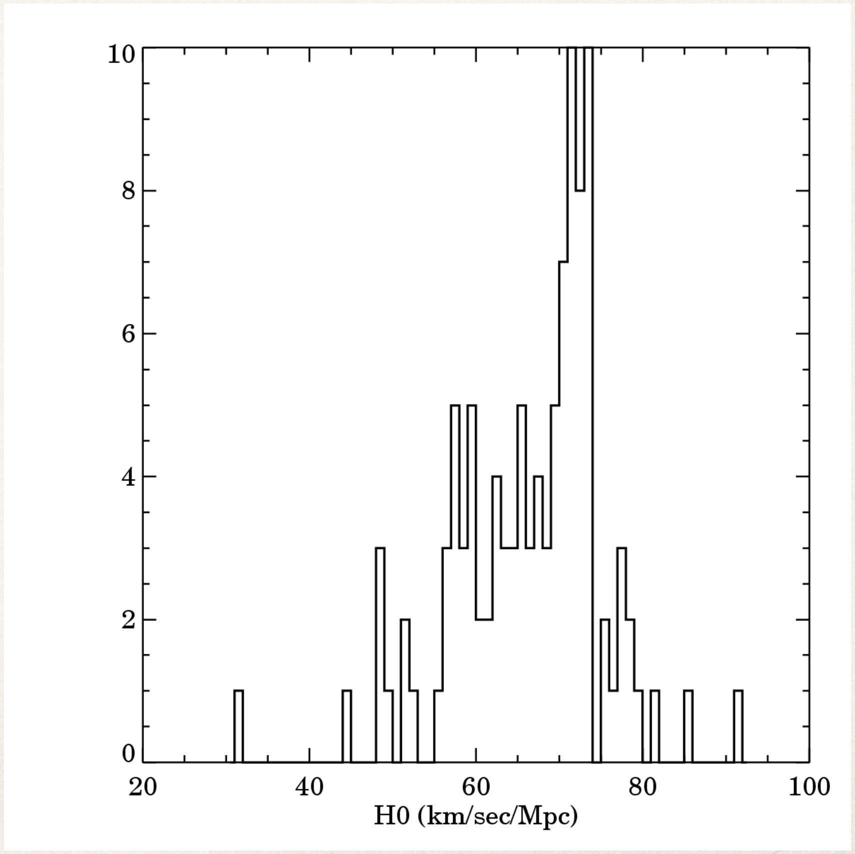
- Estimates of the W boson mass are discrepant between different experiments
- Someone must be misestimating their errors...



CDF Collaboration / Aaltonen et al. 2022

Evolution of H_0, 2000-2005





What's happened since 2005?

- We'll see for ourselves. To do that, we'll need to read in an ASCII (text format) file.
- For many file formats (including both ASCII and the astronomical standard FITS format) the **`astropy.table.Table`** class works well for I/O. To import it do:
- `from astropy.table import Table`
- The Table class can handle many file formats; go to http://docs.astropy.org/en/stable/io/unified.html#table-io for a list.
    - The **`Table.read()`** method reads a table into an astropy "record array": like a numpy array, but can include multiple data types.
    - We'll instead focus on the pandas package; format options for it may be found at https://pandas.pydata.org/pandas-docs/stable/reference/io.html

ASCII I/O in Python

1) First, look at the contents of the file. On Mac or Linux, you can do this with:

```
!cat H0_data.txt
```

The ! 'magic' command in jupyter causes anything after the ! to be executed just as if you had typed it at a terminal command prompt (On Windows, you'll need to use notepad to open the file).

2) After determining the right format, read in the file:

```
# to read in with astropy.table:
data_table=Table.read('H0_data.txt',format='ascii.tab')
# to read in with pandas:
data=pandas.read_csv('H0_data.txt',sep='\t',comment='#')
```

What is in a `pandas` Dataframe or an `astropy` Table?

- `pandas` primarily organizes data into tables with labelled rows and columns: a "Dataframe". Each column of data comprises a "Series": much like a numpy array, but labelled.
- E.g., each row could correspond to a person and each column would specify a different attribute of the person (first name, last name, etc.).
- `astropy.table` loads data into a 'recarray' object, which is similar in nature; it again provides labelled columns for each row, and again columns of the table can act like numpy arrays. You can generally convert a recarray to a pandas dataframe with the recarray's `to_pandas()` method.
- Let's see what's in data and data_table:

What is in a `pandas` Dataframe or an `astropy` Table?

1) print out the list of column names: This is an attribute of the data object: i.e. `data.columns`, not `data.columns()` . For a recarray this is `data_table.colnames`

2) print out the first record in the table: In `pandas` this is `data.iloc[0,:]` or equivalently `data.iloc[0,:]`; for a recarray it is `data_table[0]` .

`.iloc` indicates integer-based location specification -- the first coordinate slices on rows, the second slices on columns (e.g., the H0 value is column 0, or `data.iloc[:,0]`).

3) look at what is in `data` and `data_table`: both `pandas` and the `astropy Table` class provide 'pretty' output.

4) Check out the attributes and methods associated with `data` and `data_table`: type `data.` and hit the Tab key to see a list, then do the same for `data_table`.

Working with individual columns

- For convenience, we can define variables that contain arrays corresponding to individual columns of `data`; these will in many contexts work just like ordinary numpy arrays. 2 options:

```
# the following method works in both pandas and astropy
#h0 = data['h0']
#h0t = data_table['H0']
...
# this method works in pandas only
h0 = data.H0
errplus = data.error_plus
errminus=-data.error_minus
date=data.year
```

Making plots with error bars

- We can use `plt.errorbar()` to draw a plot with error bars. **Bring up the documentation on this function.**
- Now, we want to **plot H0 as a function of year using error bars**; try this with both symmetric error bars (choosing either **errplus** or **errminus**) and then modify to use asymmetric error bars. To do the latter, you provide a list of two arrays containing one error for each measurement; i.e., generically, `yerr = [minus_error,plus_error]` where minus_error is the length of the lower error bar and plus_error is the length of the upper (note that we want nonnegative numbers for both!).
- **Limit the plot range to only show results from 2000 - 2025.**

Selecting data and comparing histograms

- We want to compare the distributions of H0 values from 1999-2001 (pre-Key Project release), 2001-2012, and 2012+.
- We want to select measurements from each era and overplot their histograms on the same graph. We have multiple options for selecting subsets of the data:
- 1) If you read things in with pandas, you can use **`np.where`**() to get the indices in each array for a given date range, and then plot the corresponding values by slicing **`h0.iloc`** with those indices; e.g.:

```
wh90s = np.where( np.logical_and(date > 1990, date < 2000) )
plt.hist(h0.iloc[wh90s],bins=20)
```

Selecting data and comparing histograms

2) Alternatively, you can use a boolean array to select the rows you want from a numpy array, pandas dataframe, or series:

```
is90s = np.logical_and(date > 1990, date < 2000)
plt.hist(h0[is90s],bins=20)
```

3) If you read things in as an astropy recarray, you can use `np.where()` to get the indices, and then plot the corresponding values by just slicing the h0 values with those indices:

```
wh90s = np.where( np.logical_and(date > 1990, date < 2000) )
plt.hist(h0t[wh90s],bins=20)
```

- **Select measurements from each era and overplot histograms using different colors for each.** Use the bins= and/or range= keywords to ensure that the exact same binning is used for each sample. Be sure to include a legend! (things may be easier to see if you use histtype='step' or use the alpha keyword to make your histograms semitransparent, e.g., `alpha = 0.3`)

What's happened since 2001?

- What we've done so far:
  - Read in the file `hubble_trim.dat` using `pandas.read_fwf()`
  - Examined the resulting data table (`data`)
  - Put the columns of the data table in their own arrays, e.g.
    `h0 = data.h0`
  - Plotted H0 vs year with error bars using `plt.errorbar()`
  - Compared the distributions of H0 values from 1999-2001 (pre-Key Project release), 2001-2003 (pre-WMAP), and 2003-2010 using `plt.hist()`
- What we want to do next:
  - Derive confidence intervals for the value of H_0, derived from recent measurements

How might we turn recent measurements into a confidence interval for H_0?

- Before, we generated fake, random data from known distributions to produce Monte Carlo simulations to do things like estimate coverage of different ways of making confidence intervals.
- What if we don't know the distributions? Is there any way to make fake datasets using only *observed* data?
  - **The answer is yes!**
- Resampling methods calculate statistics from new datasets generated out of the original data.
- If the data are all independent from each other and their ordering doesn't matter, these techniques can provide reliable confidence intervals for any statistic derived from a dataset (for large *n*).

Bootstrap resampling

- The most common technique is known as ***bootstrap resampling***.
- Suppose we have N data, $x_1 \ldots x_N$.
  - A bootstrap resampling of that data consists of generating another set of data, $y_1 \ldots y_M$, randomly choosing one of the original N data points for each of the y_i.
  - Most commonly, we do this "with replacement": allowing any of the x_i to occur more than once amongst the y_i. Typically, a sample with replacement will use M=N.
    - So, with 4 data points, $x_1 \ldots . x_4$, $[x_1, x_1, x_4, x_4]$ could occur as a boostrap sample with equal probability as $[x_4, x_3, x_2, x_1]$ or any other specific choice for the 4 'new' data points.

Bootstrap resampling

- The basic idea is that we're making a Monte Carlo, but instead of drawing from some known PDF, we're using the distribution of the values in the dataset itself.

- For independent, identically distributed data with finite mean, variance, etc., this should obviously converge to giving the same result as drawing random samples from the true distribution, if n is large enough.

Bootstraps in Python

- In Python, one way to do bootstraps is by generating an array of random index numbers (i.e., index within the orignal array), and then addressing the original array we want to make bootstraps from with the array of index numbers. Their shapes need not match if we use recarrays, but in pandas this doesn't work.

```
# this code will work if h0 was derived from a recarray of is a numpy array, but
not if it is a pandas Series:
hdata=h0[np.where(date > 2001)]
ndata=len(hdata)
nbootstraps=int(1E4)
bootidx=np.floor(random.rand(nbootstraps,ndata)*ndata)
bootidx=bootidx.astype(int)
hboot=hdata[bootidx]
```

Bootstraps in pandas

For pandas Dataframes, that code won't work, but we can use other tools.

1) the `data.sample()` method: this will generate new bootstrap samples, but **can only create one-dimensional samples** (i.e., only one bootstrap sample, not many realizations).
2) the `numpy.random.choice()` function will also generate bootstrap samples from a dataframe, but can generate multi-dimensional samples!:

```
# this works in pandas:
hboot=np.random.choice(hdata,(nbootstraps,ndata) )
```

Using the results

- The array hboot contains 104 different 'new' 134-element datasets. We can estimate the PDFs for statistics of the data from their distribution amongst the bootstrap samples!
- For instance, we can predict the standard deviation of the mean:

```
err_predicted=np.std(hdata)/np.sqrt(ndata)
```

and compare to the standard deviation amongst the means or medians derived from each bootstrap sample:

```
# the slow way
means=np.zeros(nbootstraps)
for i in arange(nbootstraps):
    means[i]=np.mean(hboot[i,:])
```

Using the results

- The array hboot contains 104 different 'new' 134-element datasets. We can estimate the PDFs for statistics of the data from their distribution amongst the bootstrap samples!
- For instance, we can predict the standard deviation of the mean:

```
err_predicted=np.std(hdata)/np.sqrt(ndata)
```

and compare to the standard deviation amongst the means or medians derived from each bootstrap sample:

```
# the fast way
medians = np.median(hboot,axis=1)
print( ??? ) # mean and median of the means array
print( ??? ) # mean and median of hdata
print( ??? ) ## write code to print the std. dev. of the means and of the medians,
# and compare to the predicted error
```

Using the results

- What do the distributions look like? **Plot the histograms of both means and medians and compare (use a bin size of 0.1).**
- We can use the distribution of the bootstrap samples to approximate the true distribution of the mean, median - or anything else we calculate from a dataset
- e.g., in fitting a line, we could bootstrap amongst all the different x/y pairs, and estimate the errors in the parameters of that line from the distribution of values you get amongst the different bootstrap samples.
- So, for instance, a 95% confidence interval for the mean or median of the data would, approximately, correspond to the range from the 2.5 percentile of the mean (or median) of the bootstrap samples to the 97.5 percentile.

Sorting

- We can find the values corresponding to these percentiles only if we rank-order all of the different bootstrap results. The **`np.sort`**() function in Python sorts an array, or **`np.percentile()`** gives percentile values.
- **`np.sort(data)`** returns a sorted version of the array **`data`**; or **`np.argsort(data)`** returns a list of *indices*: the **index #** of the lowest element of **`data`** first, the next-lowest second, etc.; so:

```
sorteddata=data[np.argsort(data)]
```

will return a sorted version of the array **`data`**.

- These also can be used as methods of numpy arrays; so **`data.sort()`** sorts the array named data in-place (i.e. replaces it by a sorted version) while **`data.argsort()`** returns the indices that sort that array.

Percentiles and sorting

An example application: if we want a 95% range for the median, we could do any of:

```
print( np.percentile(medians, [2.5, 97.5]) )
```

Or:

```
sortmedians = np.sort(medians)
print( sortmedians[int(0.025*nbootstraps)],sortmedians[int(0.975*nbootstraps)] )
```

Or:

```
sortidx=np.argsort(medians)
print( medians[sortidx[int(0.025*nbootstraps)]],
       medians[sortidx[int(0.975*nbootstraps)]] )
```

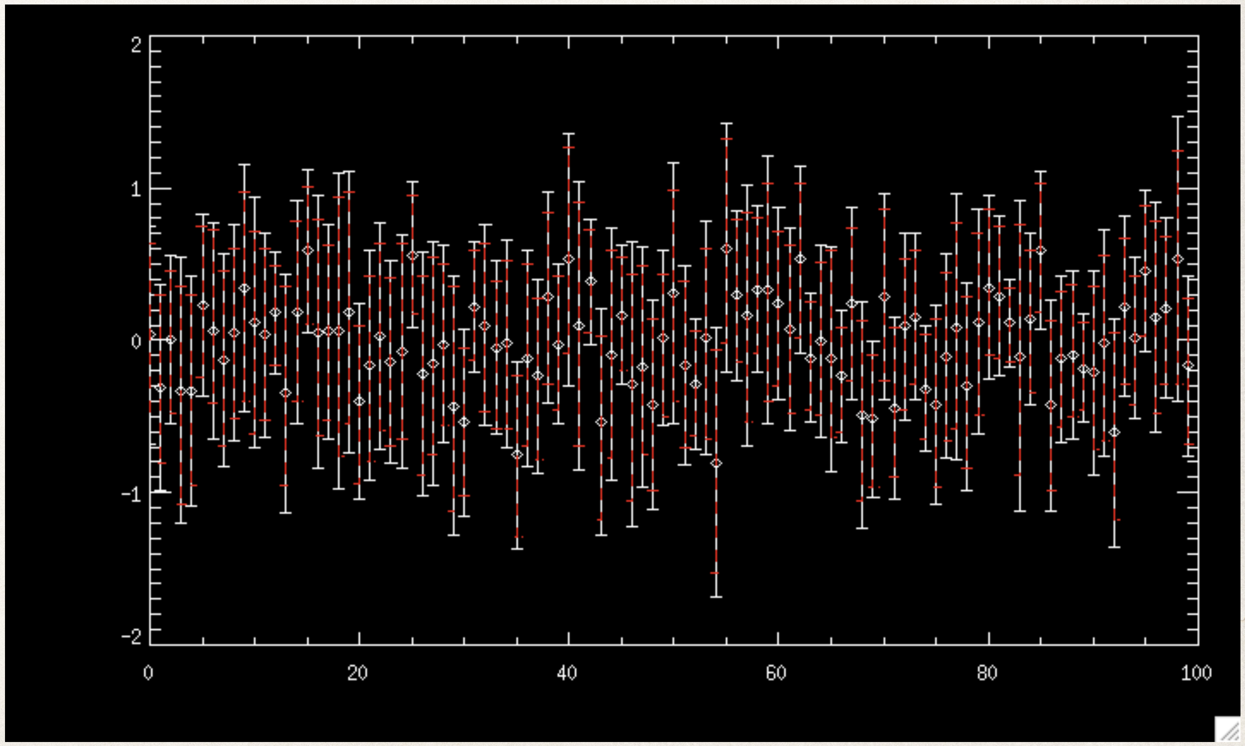
to get an approximate 95% range for the median value of recent H_0 measurements.

Confidence interval for the mean

Using one of these methods, determine a 95% confidence interval for H_0 from the <u>means</u> of each bootstrap sample.

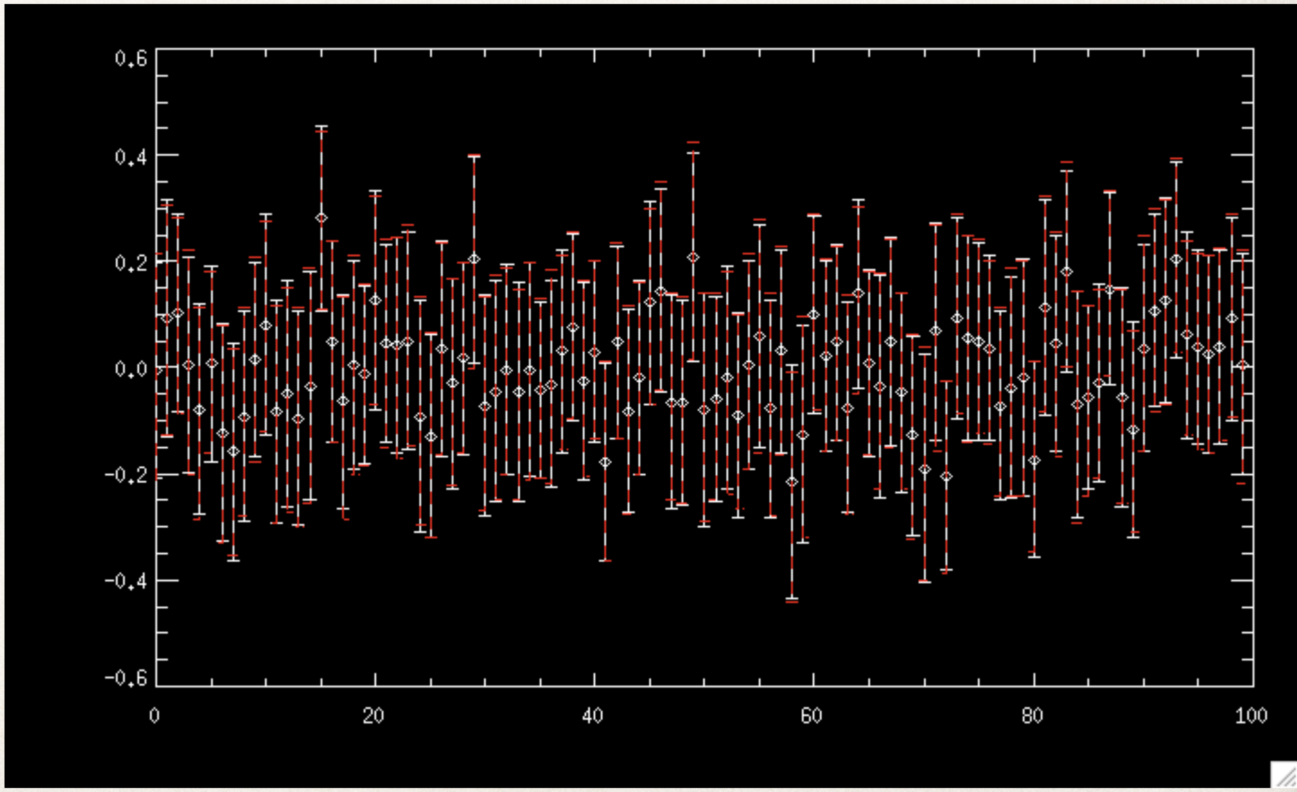
How does this compare to mean & sigma-based intervals?

- Here, I've generated 100 sets of 10 data from N(0,1) and plotted the 95% confidence interval for the mean using t statistics, as we talked about last week (these intervals extend out 2.26σ_s from the mean). Then I overplot the confidence interval from bootstrap resampling each dataset in red.
- In this situation, the simple bootstrap underestimates errors, for much the same reasons that [m-2$\hat{\sigma}_{\bar{x}}$, m+2$\hat{\sigma}_{\bar{x}}$] would.



How does this compare to mean & sigma-based intervals?

- For sets of 100 data from N(0,1) instead of 10, the bootstrap confidence intervals are almost perfect.
- Note that we didn't have to assume anything about Gaussianity for them, though!



0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
0
20
40
60
80
100