Errors and Bootstrap Resampling

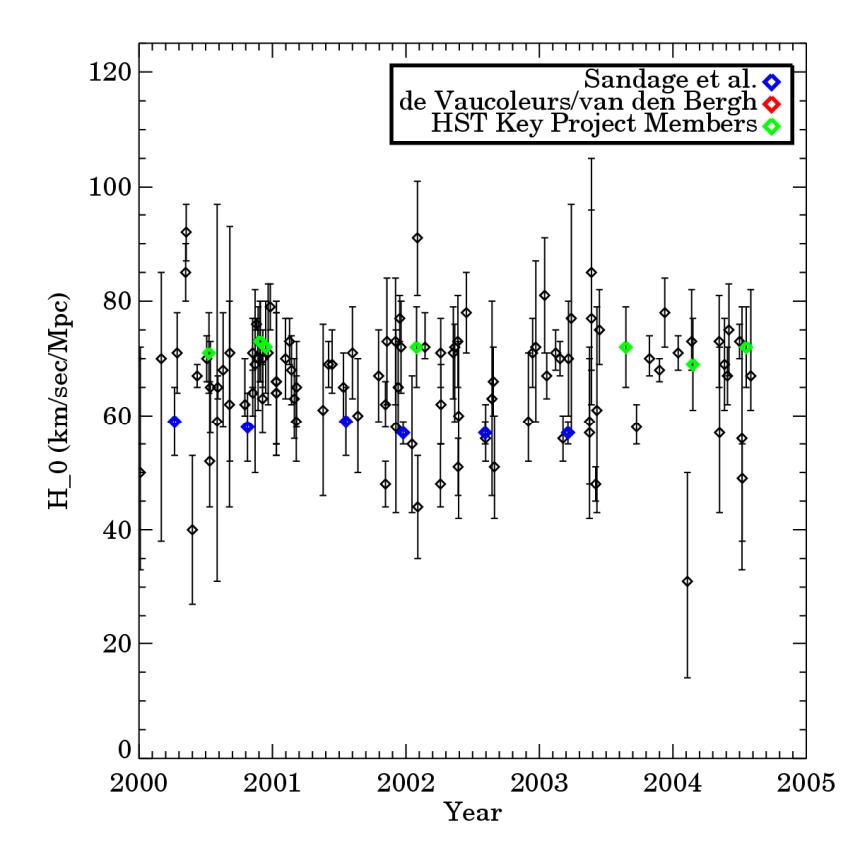
Statistics and Data Science

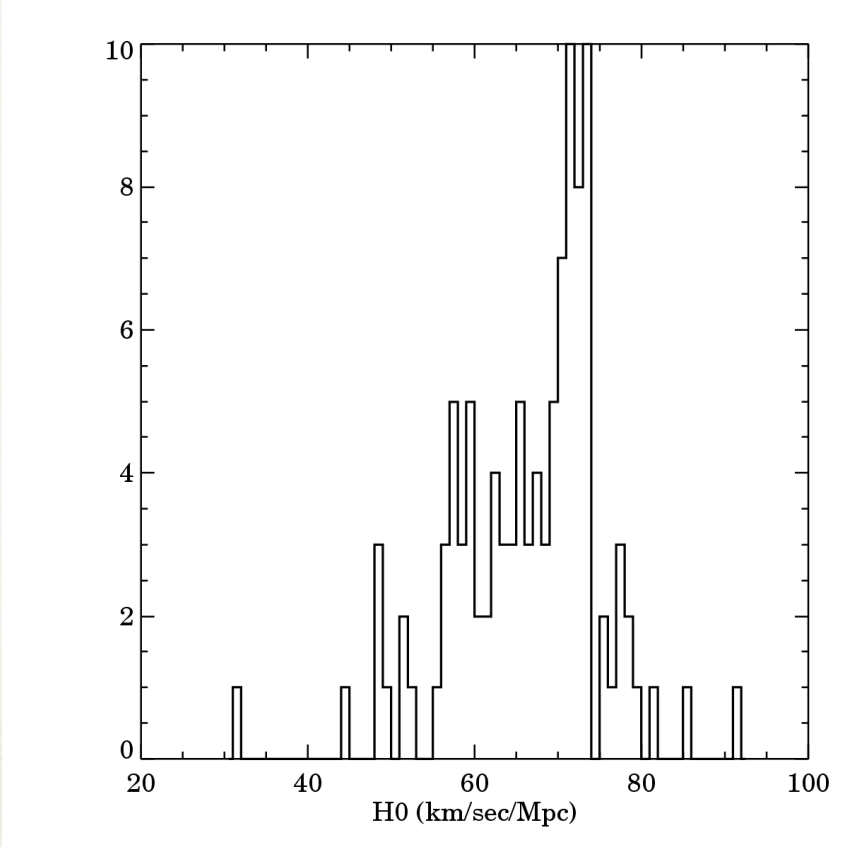
Spring 2025

Goals for today: you should be able to...

- **lecture 13/14 notebook**:
  - Obtain bootstrap samples in Python
- Propagate errors in one quantity to errors in another quantity

Review: Evolution of H_0, 2000-2005





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 H_0 vs year with error bars using `plt.errorbar()`
  - Compared the distributions of H0 values from 1999-2001 (pre-Key Project release), 2001-2012, and 2012+ 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, $x_1 ... x_N$.
  - A bootstrap resampling of that data consists of generating another set of data, $y_1 ... y_M$, 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, $x_1 .... x_4$, $[x_1,x_1,x_4,x_4]$ 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

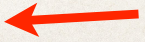
- The basic idea is that we're making a Monte Carlo, but instead of drawing from some known PDF, we're using the distribution of the values in the dataset itself.

- 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)]] )
```

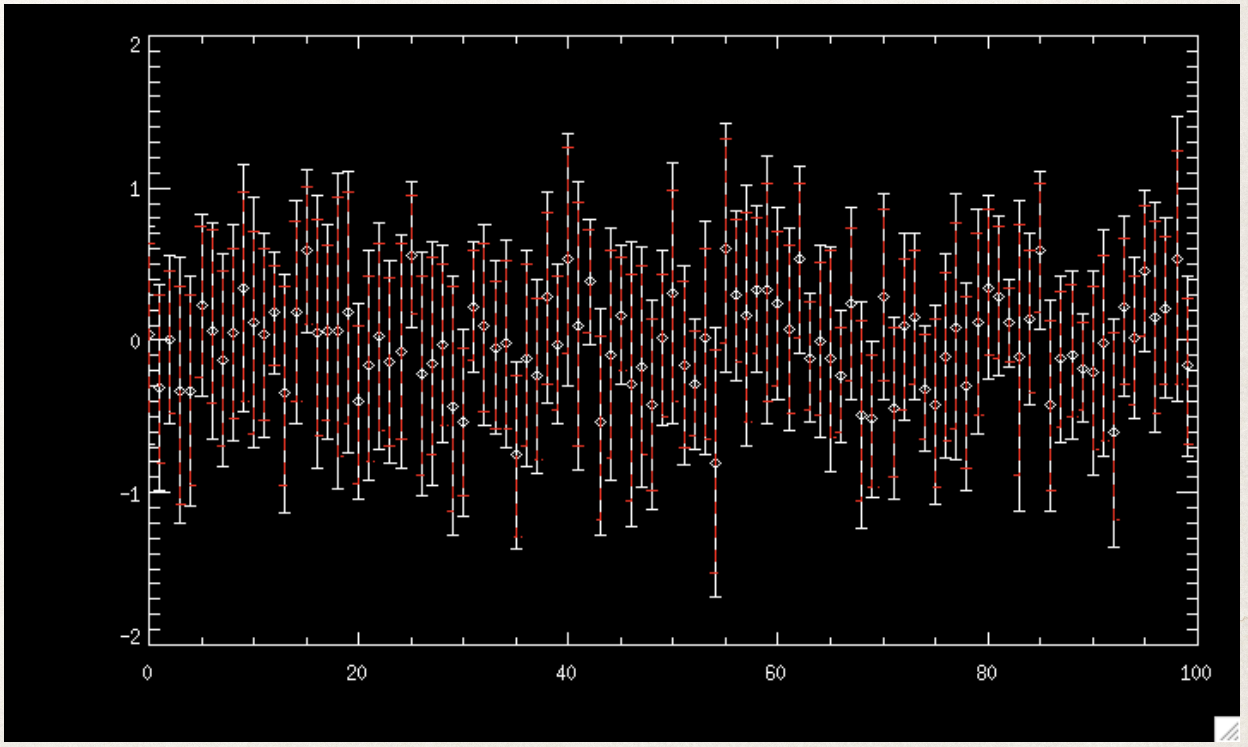
to get an approximate 95% range for the median value of recent H_0 measurements.

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.

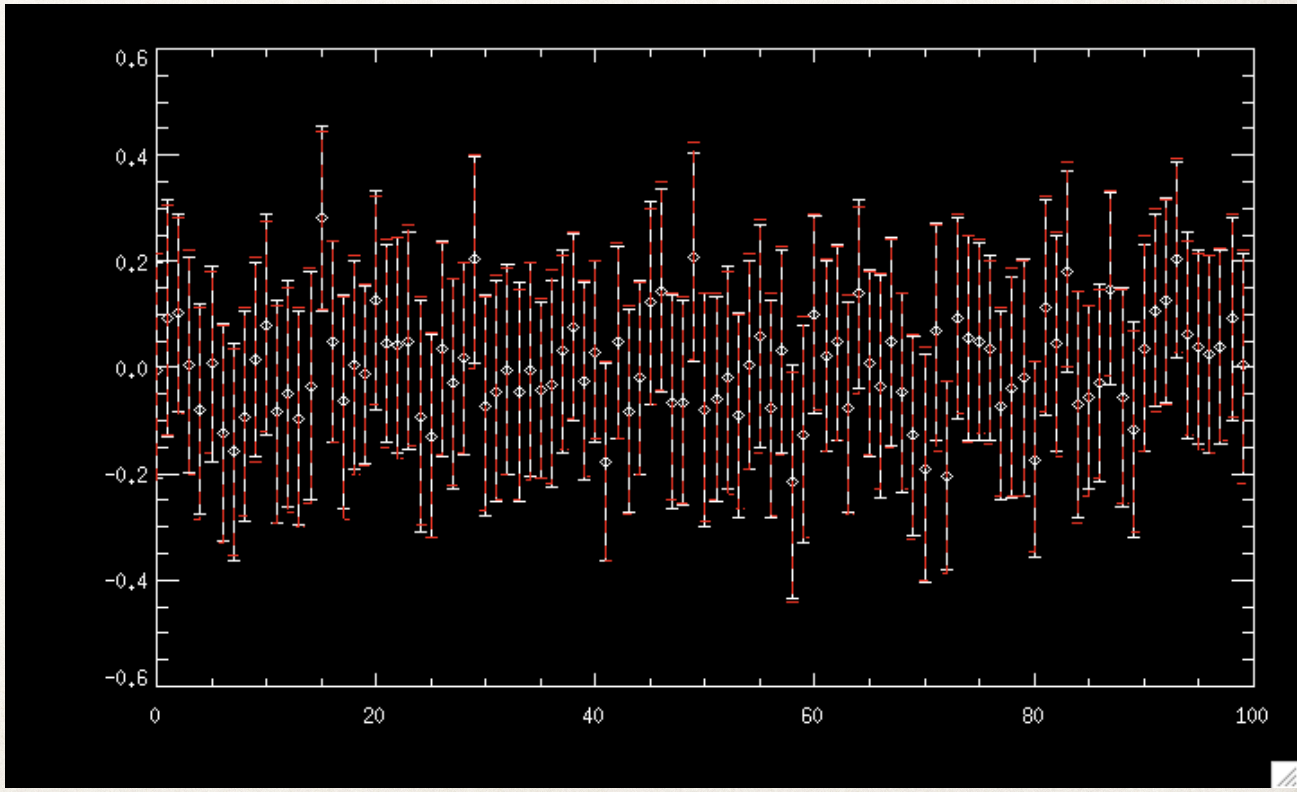
How does this compare to mean & sigma-based intervals?

- 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 $2.26\sigma_s$ 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!



Homework 4, Due Wednesday March 12

1) Go to https://docs.google.com/forms/d/e/1FAIpQLScfZ_8ilHgdsypLolGbn2K-0HAYAaBUhX3HFOXFRCd58ynH5Q/viewform?usp=header ASAP after class and enter in your height in meters (to the nearest 0.01 m). I will provide a file containing the results on Canvas.

A) Based on this data, determine 68%, 95%, and 99% confidence intervals for the average height of people in the US, and describe the meaning of that interval. Use limits determined from the *t* distribution in determining your confidence intervals.

B) Determine 68%, 95% and 99% confidence intervals for the average height of people in the US using bootstrap resampling. Assess how this compares to the intervals you derived in A.

C) The actual mean height of Americans is 1.7 m. What do you conclude about your results given this information?

Studentized bootstrapping

- For small n (and/or small numbers of bootstrap samples) the approximations we are making may be imperfect.
- The ***Studentized bootstrap*** or ***bootstrap-t*** method yields more accurate confidence intervals, especially in such cases; see:
  https://www.uvm.edu/~statdhtx/StatPages/Resampling/BootstMeans/bootstrapping_means.html
  or http://www.tau.ac.il/~saharon/Boot/10.1.1.133.8405.pdf (note: they consider only CIs that are upper limits)

Studentized bootstrapping

- We can use bootstrap samples to empirically determine the distribution of t for data that is not necessarily Normally-distributed by looking at the distribution of

  $$t^* = (\langle x_i \rangle - \langle x \rangle)/(\sigma_{s,i}/n^{1/2})$$

  where x_i and $\sigma_{s,i}$ are derived from the $i'th$ bootstrap sample and $\langle x \rangle$ is derived from the original data

- An 100-α% confidence interval then can be obtained from:

  $$[\ \langle x \rangle - t^*_{\alpha/2}\ \sigma_s/n^{1/2},\ \langle x \rangle + t^*_{100-\alpha/2}\ \sigma_s/n^{1/2}\ ]$$

  where t^*_x indicates the xth percentile of the t^* values.

- This will give confidence intervals that have more accurate coverage than if we used the t cutoffs based on the standard t distribution, if the data is non-Gaussian

Smoothed bootstraps

- When we plotted the histogram of the bootstrap medians, it was highly quantized. We can avoid this by adding a modest amount of noise to each data point in the bootstrap sample, so it does not have to match an original example exactly - enough to overcome the discreteness, but ideally small compared to the errors on each measurement.
- We can make a smoothed version of our set of bootstraps **`hboot`** by:

```
sboot=hboot+random.randn(nbootstraps,ndata)
```

which will add a random number with RMS 1 km/sec/Mpc to each value in the bootstrap.

- Then we can calculate means and medians of each bootstrap sample:

```
smeans = np.mean(sboot,axis=1)
smedians=np.median(sboot,axis=1)
```

Smoothed bootstrap results

- **Overplot histograms of the medians and smedians arrays using the same binning for each.**

- **Then, determine 95% confidence regions for the smoothed mean and smoothed median, and compare to your values from the unsmoothed bootstraps.**

Smoothed bootstrap result

- In one particular realization of a smoothed bootstrap, the 95% range for the median went from [68.9,71.0] to [68.88,70.79]
- Although the smoothed bootstrap yields much more aesthetically pleasing distributions, it often doesn't change confidence intervals much

How do our results compare to more recent values?

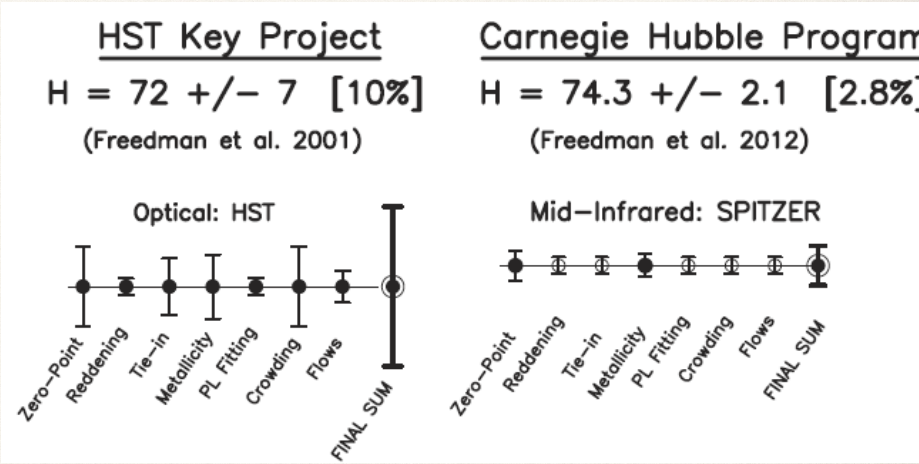
- Distance ladder via Cepheids:

74.3 +/- 2.1 (Freedman et al. 2012)

73.5 +/- 1.6 (Riess et al. 2018)

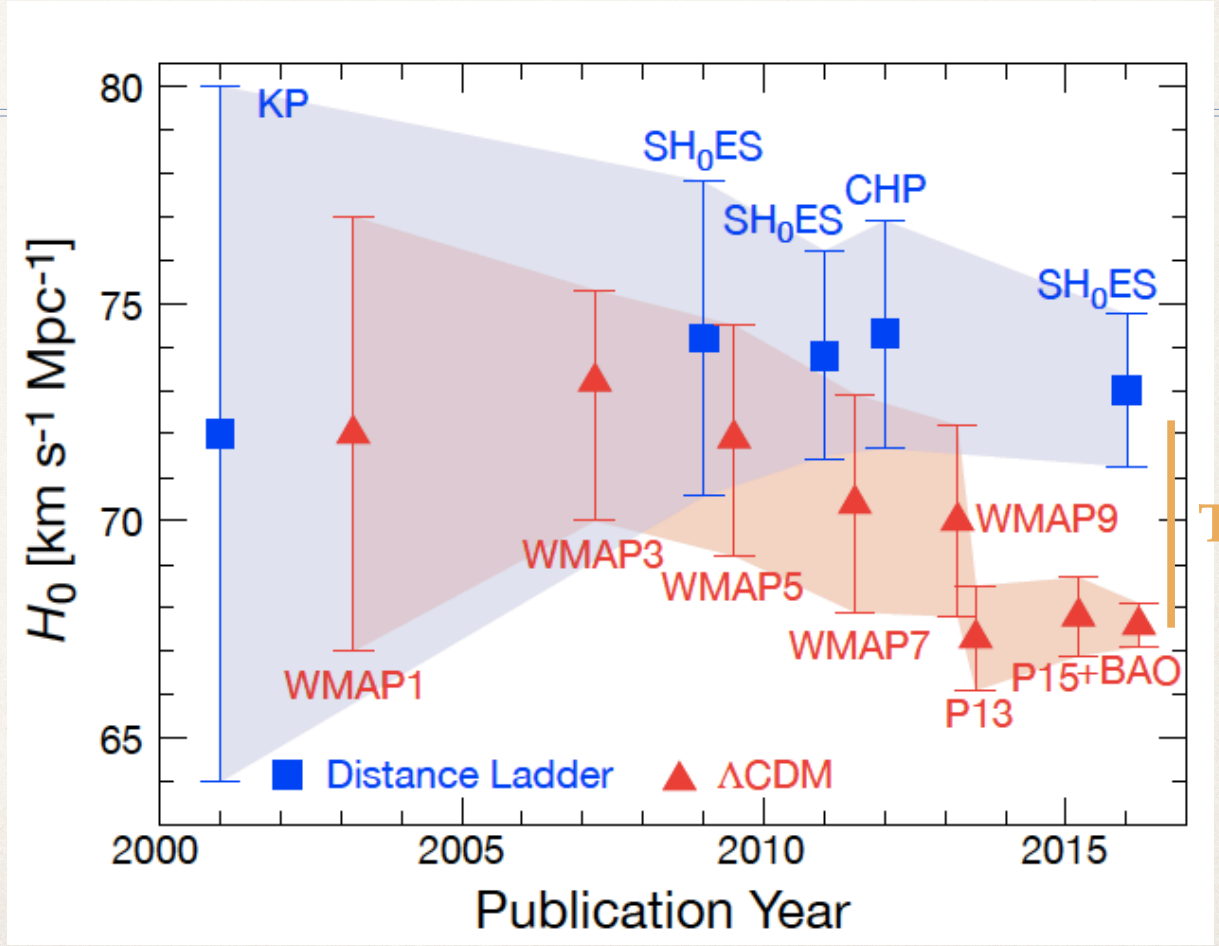
- Distance ladder via TRGB stars:

69.8 +/- 2.2 (Freedman et al. 2019)



- WMAP9 cosmic microwave background measurements+ assuming cosmological model is simple: 70 +/- 2.2
- Planck cosmic microwave background measurements: 67.4 +/- 0.5, or +0.7/-3.8 if generalize cosmology

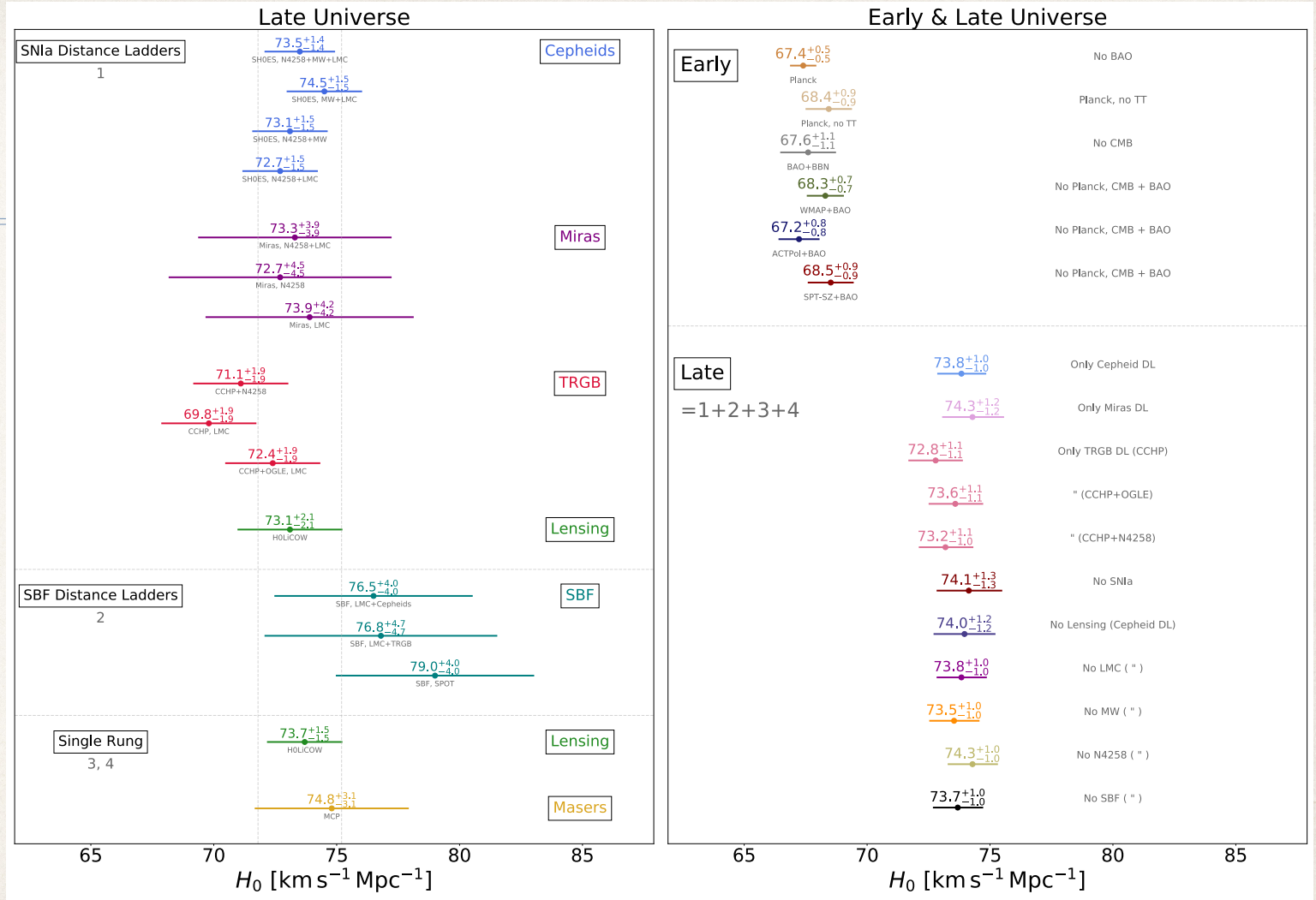
Recent measurements of H0 from distance ladder vs CMB



TRG

- Figure: W. Freedman

Recent measurements of H0 from distance ladder vs. CMB



Riess 2020, https://arxiv.org/abs/2001.03624 ; see also https://arxiv.org/abs/2203.06142

The Jackknife

- If the data are not all independent of each other the assumptions underlying bootstraps don't work
- An older resampling technique, known as the jackknife, is commonly used in that case.
  - Suppose the data can be broken up into N blocks, each of which is (mostly) independent of each other (e.g., corresponding to different regions of the sky).
  - In that case, we can leave out 1 block at a time, and calculate the mean (say) of all the remaining N-1 blocks of data. We can then estimate the standard deviation of the mean as sqrt((N-1)/N) times the standard deviation of the mean measurements from those N subsets.

Propagation of errors

- Suppose we know the uncertainty in one quantity. Can we predict the uncertainty in related quantities? This is a crucial question for experiment design!
- As an example, suppose we measure quantity x, but really want to know quantity y. How small do the errors in x need to be to make the errors in y small enough?

Propagation of errors

- Suppose we make measurements of variable x, giving us estimates of the mean & standard deviation, μ & σ, of the Normal distribution it is drawn from.

- What will be the probability distribution for $y = x + b$, where b is a constant? It must be the case that:

  $\text{mean}(y) = \boldsymbol{E}(y) = \boldsymbol{E}(x+b) = \int f(x)(x+b)dx = \boldsymbol{E}(x) + b = \mu + b$

  $\text{variance}(y) = \boldsymbol{E}((y-(\mu+b))^2 = \boldsymbol{E}((x+b-(\mu+b))^2 = \boldsymbol{E}(x-\mu)^2 = \sigma^2$

- so if x is described by $N(\mu,\sigma^2)$, then y is described by $N(\mu+b,\sigma^2)$.

Propagation of errors

- Similarly, what will be the probability distribution for y = ax, where a is a constant? It must be the case that:

  $\text{mean}(y) = \boldsymbol{E}(y) = \boldsymbol{E}(ax) = \int f(x)(ax)dx = a\boldsymbol{E}(x) = a\mu$

  $\text{variance}(y) = \boldsymbol{E}((y-(a\mu))^2 = \boldsymbol{E}(ax-a\mu)^2 = \boldsymbol{E}a^2(x-\mu)^2 = a^2\,\boldsymbol{E}(x-\mu)^2 = a^2\sigma^2$

- so: if x is described by $N(\mu,\sigma^2)$, then y is described by $N(a\mu,a^2\sigma^2)$.

- In fact, if $y=ax+b$, then if x is described by $N(\mu,\sigma)$, y will be described by $N(a\mu+b,a^2\sigma^2)$; if x is Normally-distributed, so will y be.

The more general case

- Suppose y is some function of x: $y=g(x)$.
- Then (if conditions like differentiability hold) Taylor's theorem tells us that y can be accurately described over some small interval about some value of x, x_0, as just a linear function of x :

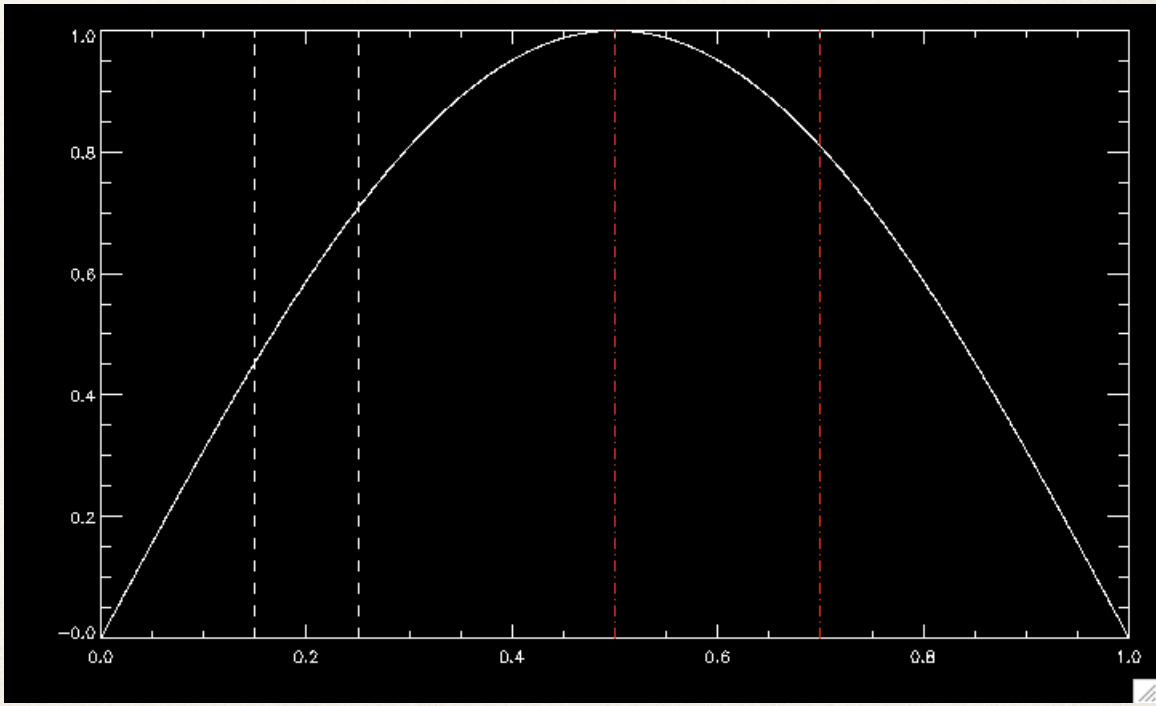
$$y \approx g(x_0) + (dg/dx)(x-x_0)$$

Then $(y-g(x_0))=y-y_0 \approx a\,(x-x_0)$; so if $x_0=\mu$, $(x-x_0)$ is distributed as $N(0, \sigma^2)$, and $y-y_0$ will be approximately described by:

$$N(0, a^2\sigma^2)=N(0, (dg/dx)^2\,\sigma^2)$$

The more general case

- Via Taylor's theorem, this will be accurate so long as $(d^2g/dx^2)\ \sigma^2$ is small - so if the 2-sigma range is the one shown by the white dashed lines, this as an excellent approximation, while if it's the red dashed lines, the accuracy will be less.



Example:

- Suppose we have estimated the brightness of a "standard candle" - an object whose luminosity we know - with 10% error and want to infer its distance. What will the fractional uncertainty in the distance be?
- brightness ∝ Luminosity / distance^2, so $d \propto b^{-1/2}$
- Let $d = a\, b^{-1/2}$. Then $\mathrm{d}d/\mathrm{d}b = -a/2\ b^{-3/2}$, so $\sigma(d) = a/2\ b^{-3/2}\,\sigma(b)$; and so

$$\sigma(d)/d = \frac{\frac{a}{2} b^{-\frac{3}{2}} \sigma(b)}{a b^{-\frac{1}{2}}} = 1/2\ \sigma(b)/b$$

- Hence, if we know b to 10%, we have determined the distance to 5%, roughly.

More generally:

- Let f be some function of a set of variables x_j.
- If all of the variables except for x_i are constant, the previous expression gives $\sigma^2(f)=(df/dx_i)^2\,\sigma^2(x_i)$
- Recall that, if x and y are independent, Normally distributed variables, $x+y$ is Normally distributed with $\sigma^2(x+y)=\sigma^2(x)+\sigma^2(y)$; so, holding one variable at a time constant, we find:

$$\sigma^2(f) = \sum_i \left(\frac{\partial f}{\partial x_i}\right)^2 \sigma^2(x_i)$$

- e.g., if $f=x_1x_2$, then $\sigma^2(f) = x_2^2\,\sigma^2(x_1) + x_1^2\,\sigma^2(x_2)$

Let's work out a more complicated case...

- The rate at which cosmic density fluctuations grow is proportional to $f = \Omega_m^{\gamma}$, where Ω_m is the density of matter in the Universe relative to the critical density at which the geometry of the Universe is flat, and $\gamma = 0.56$ (regardless of other cosmological parameters) if General Relativity is correct.
- If we make a measurement of f accurate to 10% (so $\sigma_f / f = 0.1$), what will be the uncertainty in γ (i.e., σ_γ)?
- **Solve this symbolically, then assume $\Omega_m = 0.3$ and evaluate.**

Some standard cases

- The most useful case is probably $f=x^A y^B \ldots$
- Then you can show that $(\sigma_f/f)^2=A^2(\sigma_x/x)^2+B^2(\sigma_y/y)^2+\ldots$
- Hence, in the single-variable case, the fractional error in f will be A times worse than the fractional error in x, etc. E.g. for $d \propto b^{1/2}$, we found $\sigma(d)/d = 1/2\ \sigma(b)/b$

$f = aA^{\pm b}$	$\frac{\sigma_f}{f} = b\frac{\sigma_A}{A}$
$f = a\ln(\pm bA)$	$\sigma_f = a\frac{\sigma_A}{A}$
$f = ae^{\pm bA}$	$\frac{\sigma_f}{f} = b\sigma_A$
$f = a^{\pm bA}$	$\frac{\sigma_f}{f} = b\ln a\sigma_A$

Source: wikipedia.org

Application: Error in the weighted mean

- Suppose we have a weighted mean, so $f=\Sigma\, w_i\, x_i$, where w_i is the weight applied to the ith value, x_i.
- Then by propagation of errors $(\sigma_f)^2=\Sigma w_i^2(\sigma_i)^2$
- Typically, we will use weights proportional to $\frac{1}{\sigma_i^2}$, as that is the optimal weighting for the mean of measurements with different errors (which we found before).
- For a mean, we want $\Sigma\, w_i=1$, so $w_i = \frac{\frac{1}{\sigma_i^2}}{\sum \frac{1}{\sigma_i^2}}$
- Then $(\sigma_f)^2=\Sigma w_i^2(\sigma_i)^2 \quad = \quad (\frac{1}{\sum \frac{1}{\sigma_i^2}})^2 \sum \frac{1}{\sigma_i^4}\sigma_i^2 \quad = \quad \frac{1}{\sum \frac{1}{\sigma_i^2}}$