# Machine Learning

Statistics and Data Science

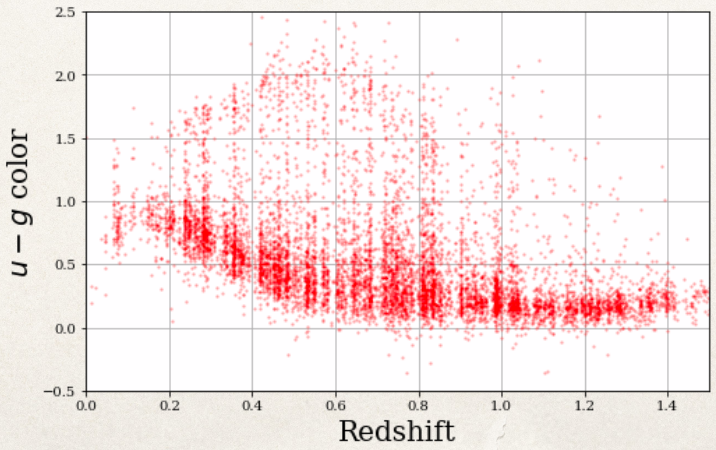
Spring 2025

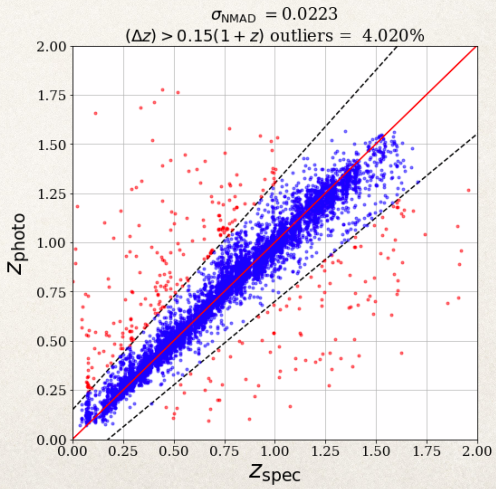
# Goals for today: you should be able to...

- **lecture 15/16 notebook**:
  - Apply machine learning methods for classification and regression
- Choose a project
- Propagate errors in one quantity to errors in another quantity

# Review: the problem we are working on: Photometric Redshifts

- Redshift ($z$) measurements allow us to determine how far back in Universe's history we are looking
- ***Photometry*** = measuring how bright something is
- ***magnitude*** is a measure of flux: e.g.: $r$ = $r$-band magnitude = -2.5 log10 ($r$ flux / $r_0$)
- In astronomy, ***color*** is the difference between magnitudes in 2 filters (with bluest specified first)
  - e.g.: $g$ - $r$ = $g$-band magnitude - $r$-band magnitude = - 2.5 log10 ($g$ flux / $r$ flux)



# Quantities we will work with

```
Full catalog:  8508  objects
['U', 'G', 'R', 'I', 'Z', 'Y', 'UERR', 'GERR', 'RERR', 'IERR', 'ZERR',
 'YERR', 'RADIUS_ARCSEC', 'ZHELIO', 'MAGB', 'UB_0']
```

- U, G, R, I, Z, and Y are measurements of magnitudes in filters (UV/ green/red/IR/ more IR/ even more IR)
- ZHELIO is a spectroscopic measurement of the redshift

$z=( \lambda_{observed}/\lambda_{restframe} ) -1$ , translated to the rest frame of the Sun (so that it does not vary annually)

  - We can ignore the other quantities today. We want to take the magnitudes, and predict ZHELIO.

- It turns out that colors are more informative about redshift than raw magnitude. We will construct *u-g, g-r, r-i, i-z,* and *z-y* colors

# Walking through the code: imports

```
from astropy.table import Table   #astropy routine for reading tables
import pandas   # we'll mostly work with things in Pandas

# Random forest routine from scikit-learn:
from sklearn.ensemble import RandomForestRegressor

# kNN routine from scikit-learn:
from sklearn.neighbors import KNeighborsRegressor

# Cross-Validation routines:
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_predict
```

# Walking through the code: reading in data

```
#read in catalog of magnitudes and redshifts
# CHANGE PATH VARIABLE TO POINT TO DIRECTORY WITH THE FILE
path='./'

# easiest way to read in a FITS BINTABLE file containing a database is
# using astropy
cat = Table.read(path+'data_trim.fits.gz')

# we then convert the catalog to Pandas form
cat=cat.to_pandas()


# How big is the catalog?
print('Full catalog: ',len(cat),' objects')

# What information does it contain for each object?
print(cat.columns)
```

- Note: Canvas changed the filename of the data file; you will need to change

```
cat = Table.read(path+'data_trim.fits.gz')
```

to

```
cat = Table.read(path+'data_trim-2.fits.gz')
```

# Walking through the code: setup for scikit-learn

```
# create convenience arrays for all magnitudes
u_mag = cat.U
g_mag = cat.G
r_mag = cat.R
i_mag = cat.I
z_mag = cat.Z
y_mag = cat.Y

#Redshift array
z = cat.ZHELIO

#vector of redshifts
data_z = z

# Now, set up input data array for scikit-learn regression algorithms
# We will include galaxy colors (expressed as differences between magnitudes
# in adjoining bands) and one magnitude.
# np.column_stack makes a 2D array out of a set of 1d arrays :
# with 6 variables we get an N x 6 numpy array out
data_colmag = np.column_stack((u_mag-g_mag, g_mag-r_mag, r_mag-i_mag,
                               i_mag-z_mag, z_mag-y_mag, i_mag))
data_colmag.shape
```

# Walking through the code: rescaling features

```
# For k-nearest neighbor we want to rescale all variables
#    to have mean 0 and variance 1
# This is often, but not always, important for machine
#    learning routines
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# the rescaling requires a fit step...
scaler.fit(data_colmag)

# then we apply the scaler to create a new array of features,
#   all normalized
scaled_colmag = scaler.transform(data_colmag)
```

- Defining a combined distance between two objects in k-nearest neighbor makes little sense if the different features have very different ranges.
- The easy way to handle this is to rescale everything to have similar span. This is often helpful or assumed to have been done when applying machine learning algorithms.
- In **`scikit-learn`**, most routines follow a standard workflow:
  - We set up an instance of a particular algorithm (object); use its **`.fit()`** method to train the algorithm; and use the **`.transform()`** or (more often) **`.predict()`** to apply the algorithm to data.

# Walking through the code: Creating instances of each algorithm

```
# We will set up an implementation of the scikit-learn
# RandomForestRegressor in an object called 'regrf'.
regrf = RandomForestRegressor(n_estimators = 50,
                max_depth = 30, max_features = 'auto')


# We need to set up an implementation of the scikit-learn
# KNeighborsRegressor in an object called 'regrn'.
# uniformly weighting 10 nearest neighbors:
regknn = KNeighborsRegressor(n_neighbors = 10, weights='uniform')

# or weighting neighbors inversely proportional to their distance:
regknn_d = KNeighborsRegressor(n_neighbors = 10, weights='distance')
```

- We can set parameters for how an algorithm is constructed when the instance of an object from that method's class is created

# Assessing performance: training vs. testing

```
if 0:
    # To better assess the quality of the Random Forest fitting,
    # we split the data into Training (50%) and Test (50%) sets.
    # The code below performs this task on the data_mags and data_z arrays:

    # 1) randomly divide the sample into 50% training and 50% testing sets
    # (e.g., data_train, z_train, and scaled_train are the training
    # portions of data_colmag, data_z, and scaled_colmag

    data_train, data_test, z_train, z_test, scaled_train, scaled_test \
           =train_test_split(data_colmag, data_z, scaled_colmag, \
                        test_size = 0.50, train_size = 0.50)
```

- Any machine learning algorithm will try to optimize performance on the particular training set it is given. This may result in overfitting.
- As a result, an algorithm may perform much better on the data it is provided as input than on new data
- We can assess the performance of an algorithm more realistically by fitting our model with one subset of the data, and testing with another.
- `sklearn.model_selection.train_test_split()` will randomly split up our data arrays into separate, matched training and test sets.

# Walking through the code: Doing the learning!

```
    #Train the regressor using the training data
    regrf.fit(data_train,z_train)

    #Apply the regressor to predict values for the test data
    z_phot = regrf.predict(data_test)
    z_spec = z_test

#Make a photo-z/spec-z plot and output summary statistics for the test set.
    plot_and_stats(z_spec,z_phot)
```

- **Evaluate the performance of random forest and kNN from the below code boxes.**
  - Activate the calculations for Random Forest and k-Nearest Neighbors in the following code boxes by changing if 0 to if 1.
- **Which algorithm does better at minimizing scatter? Reducing the fraction of outliers (objects with large deviations)?**

# Warning: do not train and test with the same data!

```
if 0:

    # use the RF regressor trained on the training set, but
    #   apply it to the training set instead of the test set
    z_phot = regrf.predict(data_train)
    z_spec = z_train

    plot_and_stats(z_spec,z_phot)
```

- **How do the statistics (NMAD + outlier rate) differ when evaluated on the training set, as compared to when using an independent test set?**

# Cross-validation

- With a 50-50 training/testing split, we are always training significantly more poorly than we would with the full dataset, and can only use half the data to evaluate how well we are doing. ***K-fold cross-validation*** provides a way around this.
- In k-fold cross-validation, we split the data into k subsets. We loop over the subsets, training with all but one and testing with the other; in the end, we get the performance of training with a fraction *(k-1)/k* of the data, but are able to get test statistics based on the entire dataset.
- This is easy to do in `scikit-learn`, but does require running the training ***k*** times ...
  - Note: we can search multi-dimensional grids of ML algorithm parameters in an automated way and optimize parameters with cross- validation with `sklearn.model_selection.GridSearchCV`; see https://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digits.html for an example.

# Walking through the code: cross-validation

```
if 0:

    # use the RF regressor trained on the training set, but
    #   apply it to the training set instead of the test set
    z_phot = regrf.predict(data_train)
    z_spec = z_train

    plot_and_stats(z_spec,z_phot)
```

- **Compare the performance (NMAD, outlier rate...) from random forest regression with 5-fold cross-validation vs. training with only 50% of the sample.**

# What does the plotting code do?

```
#This is a function that makes a plot of photometric redshift
#    as a function of spectroscopic redshift
#    and calculates key statistics.  It will save us a lot of work.
def plot_and_stats(zspec,zphot):

    x = np.arange(0,5.4,0.05)

# define differences of >0.15*(1+z) as non-Gaussian 'outliers'
    outlier_upper = x + 0.15*(1+x)
    outlier_lower = x - 0.15*(1+x)

    mask = np.abs((z_phot - z_spec)/(1 + z_spec)) > 0.15
    notmask = ~mask

#Standard Deviation of the predicted redshifts compared to the data:
    std_result = np.std((z_phot - z_spec)/(1 + z_spec), ddof=1)

#Normalized MAD (Median Absolute Deviation):
    nmad = 1.48 * np.median(np.abs((z_phot - z_spec)/(1 + z_spec) -np.median((z_phot - z_spec)/(1 + z_spec))))

#Percentage of delta-z > 0.15(1+z) outliers:
    eta = np.sum(np.abs((z_phot - z_spec)/(1 + z_spec)) > 0.15)/len(z_spec)

    #Median offset (normalized by (1+z); i.e., bias:
    bias = np.median(((z_phot - z_spec)/(1 + z_spec)))
    sigbias=std_result/np.sqrt(0.64*len(z_phot))
```

# What does the plotting code do?

```
# make photo-z/spec-z plot
plt.figure(figsize=(8, 8))

#add lines to indicate outliers
plt.plot(x, outlier_upper, 'k--')
plt.plot(x, outlier_lower, 'k--')
plt.plot(z_spec[mask], z_phot[mask], 'r.', markersize=6,  alpha=0.5)
plt.plot(z_spec[notmask], z_phot[notmask], 'b.',  markersize=6, alpha=0.5)
plt.plot(x, x, linewidth=1.5, color = 'red')
plt.title(f'NMAD:  {nmad:6.4f} Delta z >0.15(1+z) outlier rate:{eta*100:6.3f} %', fontsize=18)
plt.xlim([0.0, 2])
plt.ylim([0.0, 2])
plt.xlabel(r'$z_{\mathrm{spec}}$', fontsize = 27)
plt.ylabel(r'$z_{\mathrm{photo}}$', fontsize = 27)
plt.grid(alpha = 0.8)
plt.tick_params(labelsize=15)
plt.show()
```

# Warning: ML methods extrapolate poorly!

```
if 0:
    # split the sample at the median magnitude; note that
    #      smaller magnitude means brighter!
    is_bright = r_mag < 23.15

    # we can use a logical statement as a mask to select only
    #       those array elements where it is true
    data_bright = data_colmag[is_bright]
    z_bright = data_z[is_bright]

    # or negate it to select where it is false
    data_faint=data_colmag[~is_bright]
    z_faint = data_z[~is_bright]

    #train the regressor with the bright data
    regrf.fit(data_bright,z_bright)

    # run on the faint test set
    z_phot = regrf.predict(data_faint)
    z_spec = z_faint

    plot_and_stats(z_spec,z_phot)
```

- In the notebook we present a variety of scenarios in which the training & test data differ in brightness, color, and redshift.
- **Assess: which of these causes the worst problems / worst predictions?**

# Random Forest for classification

```
if 0:
    from sklearn.ensemble import RandomForestClassifier

    # set up the classifier object
    classrf = RandomForestClassifier(n_estimators=50)

    # fit a classifier intended to separate objects at
    #       redshift > 0.75 from those at < 0.75
    classrf.fit(data_train,z_train > 0.75)

    # predict the classifications for the test set
    class_predict = classrf.predict(data_test)

    # test which objects are selected as being at high redshift
    ishiz = class_predict == True

    #plot redshift histograms for each sample
    bins = np.linspace(0,1.5,150)
    a,b,c=plt.hist(z_test[ishiz],bins=bins,histtype='step',label = 'High z')
    a,b,c=plt.hist(z_test[~ishiz],bins=bins,histtype='step',label = 'Low z')
    plt.xlabel('Redshift')
    plt.legend()
```

- Classification with machine learning techniques in **`scikit-learn`** works much like regression; but the target array will consist of True and False, instead of a continuous variable.

# Optimizing parameters of a scikit-learn algorithm

```
from sklearn.metrics import mean_squared_error,median_absolute_error
if 1:
    ntree_test = 10,25, 50,100
    mse=np.copy(ntree_test)*0.
    mad=np.copy(ntree_test)*0.

    for i,n in enumerate(ntree_test):
        # define the RF object with our choice of number of trees
        regrf = RandomForestRegressor(n_estimators = n, \
                                      max_depth = 30, max_features = 'auto')
        # do training AND prediction on the whole sample
        #     using cross-validation
        predicted = cross_val_predict(regrf,data_colmag,data_z, cv=5)
        #calculate mean squared error
        mse[i]=mean_squared_error(data_z,predicted)
        #calculate MAD
        mad[i]=median_absolute_error(data_z,predicted)

    plt.plot(ntree_test, mse,label='MSE')
    plt.plot(ntree_test, mad,label='NMAD')
    plt.legend()
    plt.xlabel('Number of trees')
```

- In general, it is best to test whether your results could be improved by changing the basic free parameters of the algorithm. You do need to decide what kind of loss you want to optimize though...

# Things to keep in mind when applying machine learning methods

- Machine Learning algorithms can be very good at handling data like that which they were trained on, but can extrapolate very poorly
  - In the notebook, you can see what happens when you test with data that is identical to the training set, and also for data that is systematically different in at least some way
  - Always be sure to think about whether the data you will be applying the algorithm to will really match the training data!
- Imbalanced training sets can also be an issue: if 99.9% of the training data is in one class, always outputting that class as the solution could yield a small loss and get favored...

# Projects

# Projects

- As you may recall (see lecture 1), 40% of the course grade is based on a project, which you will perform through the rest of the semester, write a few pages about, and present to the class.
- Below is a list of some project ideas. Please talk to me ASAP if you are taking the class for a grade and none of these appeal to you.
- Please use the form that I will post on Canvas to send me a list of who you intend to work with (if you have already found a partner); a ranked list of 5 projects which you are interested in; as well as a list of what afternoon times you would be free to meet next week.
- In general, I expect people will work on projects in pairs.

# DESI-based Projects

A. Develop a machine-learning based emulator that can predict stellar mass estimates as measured from DESI spectra via the `fastspecfit` software, from redshift and color / brightness measurements alone

B. Explore the use of autoencoders or other dimensionality reduction methods (UMAP or GPLVM) to describe DESI galaxy spectra with only a small set of values (as an alternative to principal component analysis).

- For larger groups: an augmentation would be to use ML to predict physical properties from the encoded values and compare to predicting from the spectra themselves

C. Test for correlations between DESI redshift distributions and the night sky emission spectrum or night sky absorption spectrum to improve large-scale-structure measurements

# DESI-based Projects

D. Test code to average together ("stack") DESI spectra. Use this code to investigate outflows from rapidly star-forming galaxies at z ~1.5 and their dependence on galaxy properties

E. Test Khederlarian et al. code for predicting emission line strengths using stacked DESI spectra or JWST high-z galaxy spectra (Marcos Tamargo-Arizmendi)

# Projects with students interested in them

F. Attempt to predict infrared emission from DESI quasars from their observed spectra, in order to search for dust-extincted quasars (Jake McGee)

G. Test correlations of supernova brightness residuals with galaxy properties from DESI (Mykola Chernyachevskyy)

H. Implement jackknife error estimation for correlation function measurements in LSST simulations (Yoki Salcedo)

I. Estimate errors on delay time distributions for transient events (like supernovae) via bootstrap / jacknife sampling of spatial pixels (Cullen Abelson)

# Projects with students interested in them

J. Develop fitting (e.g. via simulation-based inference or other MCMC-like method) for mass-size relations, incorporating the correlated uncertainties in mass and size due to photo-z errors (Yunchong Zhang)

K. Predict redshift / stellar mass / star formation rate from JWST many-band imaging using machine learning algorithms (Nathalie Chicoine, Lauren Elicker)

- Augmentation if more students are involved: compare results if predict directly from observed flux measurements vs. predicting from UMAP-compressed version of SED space

L. Apply goodness-of-fit tests (to check whether the fits are consistent with the data) and model selection tests (to assess whether models with more parameters are delivering enough better fits to be worth the complexity) to Prospector fits to star-forming clumps in JWST IFU spectra (Julissa Sarmiento)

# 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, $\mu$ & $\sigma$, 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 :

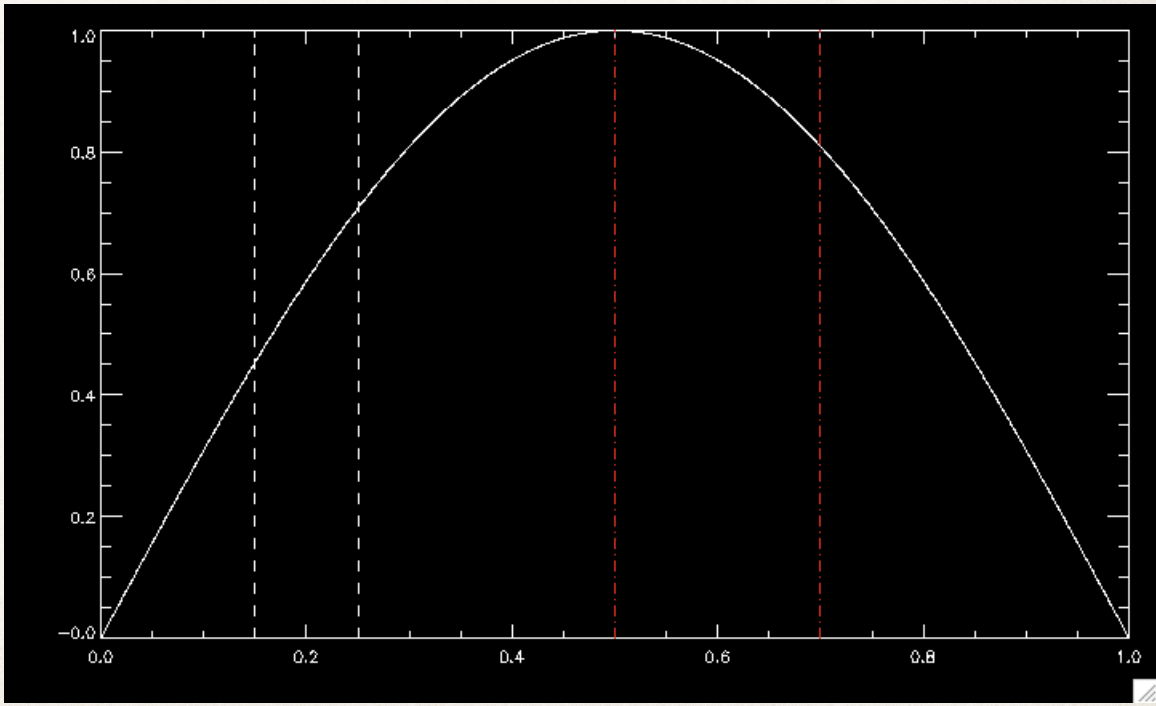
$$y \approx g(x_0) + (dg/dx)(x-x_0)$$

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 $\Omega_m$ 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 $\gamma$ (i.e., $\sigma_\gamma$ )?
- **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 $i$th 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 $w_i = \frac{\frac{1}{\sigma_i^2}}{\Sigma \frac{1}{\sigma_i^2}}$
- Then $(\sigma_f)^2=\Sigma w_i^2(\sigma_i)^2 \quad = \quad (\frac{1}{\Sigma \frac{1}{\sigma_i^2}})^2 \sum \frac{1}{\sigma_i^4}\sigma_i^2 \quad = \quad \frac{1}{\Sigma \frac{1}{\sigma_i^2}}$

Note that, if all the $\sigma_i=\sigma$, this gives $(\sigma_f)^2=\sigma^2/n$