Hypothesis testing and Maximum Likelihood

Statistics and Data Science

Spring 2025

Goals for today: you should be able to…

- Select the right hypothesis test for a problem
- Describe the advantages of maximum likelihood methods
- Explain how the Fisher information matrix can be used to evaluate constraints when experiments are combined
- **Lecture 21/22 notebook**:
    - Investigate how constraints from multiple experiments combine via the Fisher information matrix
    - Apply maximum likelihood methods for parameter determination & fitting

How to choose a test?

- The best thing to do is to apply a set of tests to fake Monte Carlo or simulated datasets similar to what you think your distributions may be, with actual sample sizes, and see what gives the best (most robust/highest-power) results
- It's always tempting to throw the whole toolkit of possible statistics at your data, looking for one that gives a significant result. If you do 20 independent tests, you'd expect one to give you a statistically significant result at the α=0.05 level, though.
- If you do want to apply multiple tests, you need to be sure to apply a Bonferroni correction or apply false discovery rate methods, to prevent an excess of Type I errors (false positives)

Modeling

- We often wish to produce mathematical descriptions of a dataset: e.g., quantify the PDF for some parameter or the relationship between multiple parameters.
- This is the problem of *modeling*: finding some function, of unknown form or parameters, that can explain the data
- Generally, we will guess at the functional form (e.g., based on a plot of the data or knowledge of physics), which will have some limited number of free parameters, which we will then determine values of/confidence intervals for using the data.

Maximum Likelihood Methods

- Suppose we have a set of *n* data, and want to determine the parameters of a model that describes them.
- In the Bayesian formalism, we can determine the PDF of those parameters by applying Bayes' theorem:

$$prob(params \mid data) = prob(data \mid params)\, prob(params) / prob(data)$$

- In the Frequentist formalism, it is inappropriate to talk about the PDF of a parameter; instead, we are concerned with how likely we are to get the observed data, *given* the assumption that the parameters have some value.
- Still, a particular statistic - the value of the parameters which maximizes the likelihood, *prob(data | params)* - has some very desirable properties.

Note: Maximum a Posteriori (MAP)

- While we'll concentrate on Maximum Likelihood Estimators, there is a Bayesian equivalent: the *Maximum a Posteriori* (MAP) estimator, i.e., the value of *params* which maximizes the posterior (and so maximizes

  prob(data | params) prob(params)

).

A previous application of Maximum Likelihood methods

- Suppose we have several independent measurements, $x_1, x_2 \ldots x_N$, which we know are all drawn from the same Normal distribution, $x_i \sim N(\mu, \sigma^2)$ with known σ; and we want to determine what our best guess at the value of μ is. We have:

$$prob(params \mid data) = prob(data \mid params)\, prob(params) / prob(data)$$

with:

$$prob(params) = 1$$

and

$$prob(data \mid params) = \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}$$

Maximizing the likelihood

- Then: *prob(params | data)* $\propto \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}$
- To maximize the posterior (i.e., choose the value of μ with greatest probability), we can just maximize the likelihood,

$$L \propto \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}$$

- Note that *ln y* is a strictly increasing function of *y*; i.e., the bigger *y* is, the bigger *ln y* is. So the value of μ which maximizes L maximizes *ln L* as well.

$$\ln L = \sum_i -\frac{(x_i-\mu)^2}{2\sigma^2} + C$$

At a maximum, $\partial \ln L / \partial \mu = 0$, so:

$$\sum_i \frac{2(x_i-\mu)}{2\sigma^2} = 0$$, so $N\mu = \sum x_i$

Advantages of Maximum Likelihood Methods

- If some commonly met conditions hold, the set of parameters (hereafter $\boldsymbol{\vartheta}$) which maximize the likelihood will:
  - 1) be asymptotically unbiased, i.e. not biased from the true value of $\boldsymbol{\vartheta}$ **if** we have a large enough sample
    - This does *not* mean that the MLE of $\boldsymbol{\vartheta}$ is unbiased for small n. E.g., the MLE for the variance of a Gaussian distribution is $1/n \sum (x_i - \langle x \rangle)^2$, which is biased compared to σ_s^2 (which uses $1/(n-1)$)
  - 2) have smaller variance than any other estimator as $n \Rightarrow \infty$; hence, it has maximum asymptotic relative efficiency (ARE)

Advantages of Maximum Likelihood Methods

- 3) As $n \Rightarrow \infty$, the distribution of the MLE approaches a multi-dimensional Gaussian with mean at the true value of the parameters (so it is *consistent*) and with covariance matrix equal to the inverse of the Fisher Information (or Fisher) matrix $I(\boldsymbol{\vartheta})$, which is:

  $I = \mathbf{E}\left((\partial\, ln\, L / \partial\, \boldsymbol{\vartheta})^2 \mid \boldsymbol{\vartheta} \right)$

  where $L(\mathbf{x},\boldsymbol{\vartheta})$ is the likelihood for data $\mathbf{x}$ and parameters $\boldsymbol{\vartheta}$, and the expectation value is evaluated over all possible data

  e.g., $I_{ij} = \mathbf{E}(\partial\, ln\, L/\partial\vartheta_i\, \partial\, ln\, L/\partial\vartheta_j) = - \mathbf{E}(\partial^2\, ln\, L/\partial\vartheta_i\, \partial\vartheta_j)$

What is required for those advantages?

- **1)** The first and second derivatives of *ln L* must all be defined
- **2)** **E** $(\partial \ln L/\partial \vartheta)=0$, where the expectation is calculated over all ϑ
  - This is equivalent to just requiring that the order of differentiation vs. integration is interchangeable, which requires that the derivative is well-defined and the density is nonuniform

Covariance matrices

- We previously defined the covariance matrix:

$$C_{ij} = \mathrm{cov}[z_i, z_j] = \mathbf{E}\left((z_i - \mu_i)(z_j - \mu_j) \right)$$

- By comparison, the variance is $\mathbf{E}\left((z_i - \mu_i)^2 \right)$, so the *i,i* element of the covariance matrix is the variance in z_i; *i.e.*, $\mathrm{cov}[z_i, z_i] = \sigma_i^2$.
- The expectation value is calculated over the probability distribution of interest; $\mathbf{E}\, g(x) = \int g(x)\, p(x)\, \mathrm{d}x$
- The minimum-variance estimator of cov[x,y] from a set of n paired measurements of x and y is

$$\mathrm{cov}\,(x,y) = 1/(n-1) \sum (x_i - \langle x \rangle)(y_i - \langle y \rangle)$$

Covariance matrices

- Any 2 independent variables have 0 covariance.
- We can also define a *correlation coefficient* $\varrho = \mathrm{cov}[x_i, x_j]/(\sigma_i \sigma_j)$, which can range from -1 (perfectly anticorrelated) to 1 (perfectly correlated; 0 if uncorrelated)
  - If you want to test if two variables are correlated, a good **robust** technique is the Spearman (or Kendall) ranked correlation coefficient, `scipy.stats.spearmanr` in Python; permutation tests also work
  - I generally only use Spearman, not the standard (Pearson) correlation coefficient, as non-Gaussianity is common

Covariance matrices & multivariate Gaussians

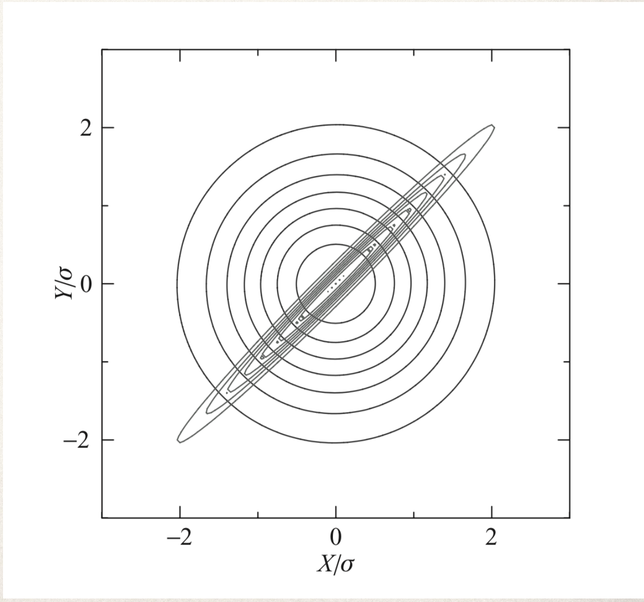
- If **x** is a vector containing the values of the *n* random variables x_i, and **μ** is a vector of the expected mean of each variable μ_i, then **x** is described by a multivariate Gaussian distribution if we can write its PDF $f(\mathbf{x})$ as:

  $f(\mathbf{x}) = (2\pi)^{-n/2} \mid \det C \mid^{-1/2} \exp(-1/2\ (\mathbf{x}^t - \boldsymbol{\mu}^t) C^{-1} (\mathbf{x} - \boldsymbol{\mu}))$

- Note: to do matrix multiplication in Python 3.5+, the syntax is `x @ y`

- Contours of constant probability from $f(\mathbf{x})$ will look like ellipses in any x_i - x_j plane

- The probability that $(x^t - \mu^t) C^{-1} (x - \mu)$ has value Δ will be given by a χ^2 distribution with *n* degrees of freedom



Useful things to remember

- If certain, common requirements hold, it is possible to perform a linear transformation on $\boldsymbol{x}$ to a new set of variables **x'** which will have a diagonal inverse covariance matrix (i.e., which diagonalizes C^{-1})
- In that case, $f(\boldsymbol{x}') \propto \prod_i \frac{1}{\sqrt{2\pi\sigma_i'^2}} \mathrm{e}^{\frac{-(x_i' - \mu_i')^2}{2\sigma_i'^2}}$
- *Principal component analysis* is a technique which will determine that transformation, yielding a new set of variables to describe the problem that are all independently distributed, **not** correlated
- This allows us to use some results proven for independent variables even for covariant cases

How ideal are ML estimates?

- We'd like our statistics to be *unbiased* - i.e., to have an expectation value equal to the parameter of interest, not offset from it.

  ✓ **ML estimates are asymptotically unbiased, but may be significantly biased for small n**

- 2) We'd like our statistics to be *consistent* - i.e., to approach the correct value of some parameter for large N. An unbiased statistic is always consistent.

  ✓ **ML estimates are generally consistent.**

How ideal are ML estimates?

- 3) A statistic should be *impartial*: our conclusions should not depend on swapping the labels on the points/datasets (unless time is an important variable) or the units used.

    ✔ **ML estimates treat all data equally**

- 4) We'd like our statistics to be *efficient* - to require as small a sample as possible to yield an accuracy/variance within some threshold.

    ✔ **ML estimates generally are optimally efficient for large n**

How ideal are ML estimates?

- 5) A statistic should have *closeness*: i.e., give a value as close as possible to the true value. We can generalize this concept to say that a statistic should *minimize loss*, where the "loss" is the expectation value of some function over all possible samples.
  - Estimators derived from maximum likelihood for a Normal distribution are equivalent to minimizing a loss given by:

$$\frac{\chi^2}{2} = \sum_i \frac{(x_i - \mu_i)^2}{2\sigma_i^2}$$

? **ML estimates minimize some sort of loss-function, though perhaps not the one we are interested in**

How ideal are ML estimates?

- 6) Ideally, a statistic should be *robust*: i.e., give the correct answer even if we have a non-Normal distribution (e.g., a Gaussian plus outliers). Although the ordinary mean has a high efficiency for normally-distributed data, it is *not* robust.

✖ **ML estimates are often not robust - they depend on understanding all the distributions involved to formulate them**

Fisher matrices

- For large enough n, the Fisher information matrix ***I*** is just the inverse of the covariance matrix of the parameters fit via ML (use `scipy.linalg.inv` or related routines in Python to invert)
- The larger the Fisher information, the smaller the variance about the MLE - so the more strongly peaked the distribution of MLEs about the true value will be
- The Fisher matrix for an experiment determines how strongly it will constrain parameters.
- If we combine results from 2 independent experiments x and y, **the information adds**: $I_{x+y}=I_x + I_y$
- The Fisher matrix is the expectation value of the Hessian matrix described in the book

Predicted uncertainty and the Fisher matrix

- We saw before that the maximum-likelihood estimator for the mean parameter of a Gaussian is the usual mean statistic. What is the expected uncertainty in μ_L?
- For 1 parameter, the covariance matrix (which is the inverse of the Fisher information) is just the variance in that parameter, $C = \sigma_i^2$; so we expect the variance $V = C = 1/I = 1/E(\partial^2 \ln L / \partial \mu^2)$
- Since $\ln L = \sum_i -\frac{(x_i-\mu)^2}{2\sigma^2} + K$,

$$\frac{\partial^2 \ln L}{\partial \mu^2} = \frac{\partial}{\partial \mu} \sum \frac{-2(x_i-\mu)}{2\sigma^2} = \sum \frac{2}{2\sigma^2} = \frac{n}{\sigma^2}$$

- The expectation value of the quantity n / σ^2 over all possible datasets will still be n / σ^2
- Hence, the variance of the estimated mean $V = \sigma^2 / n$: this is exactly what we got before.

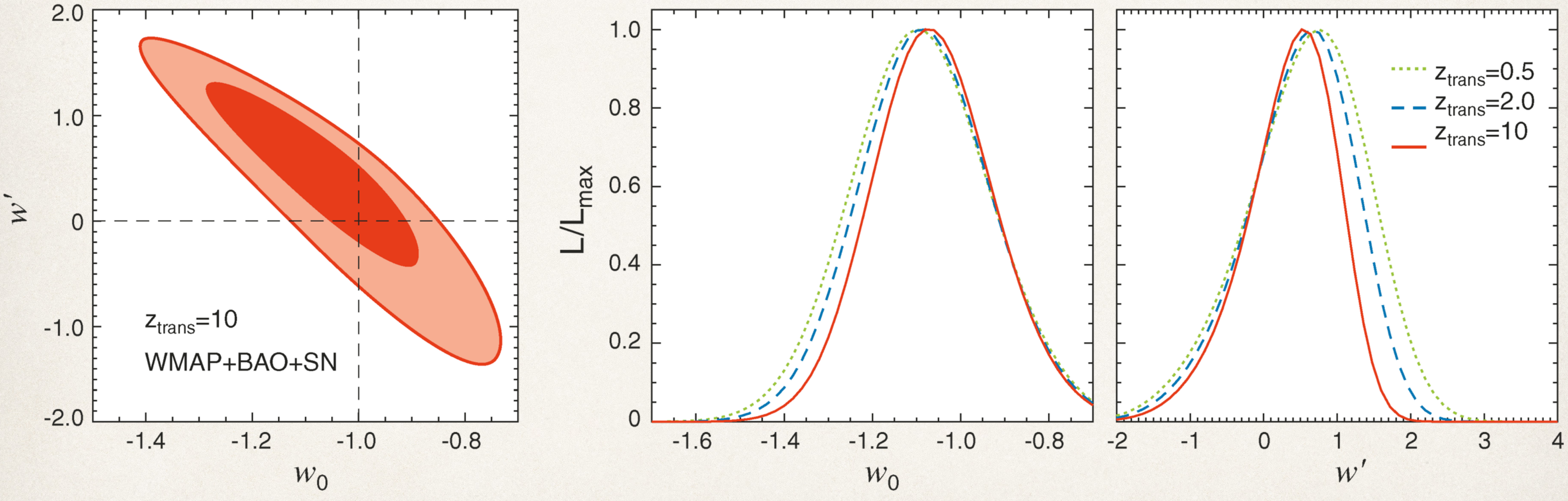
Confidence intervals from ML

- One option is to define a *credible interval* within the Bayesian formalism; e.g., the smallest region that contains a fraction $(1-\alpha)$ of the likelihood
- Another option: for large enough n, **or** if all variables are described by Normal distributions, the PDF for the ML estimate of the parameter vector will be described by a multivariate Gaussian (peaking at the true value).
  - Let's call our vector of ML estimates of the n parameters $\boldsymbol{t}$, while the vector of true values is ϑ.
  - In that case, $\ln L = -1/2\, (\mathbf{t}^t - \boldsymbol{\vartheta}^t)\, C^{-1}\, (\mathbf{t}-\boldsymbol{\vartheta}) + K$, for some constant K .
  - As we saw before, $(\mathbf{x}^t - \boldsymbol{\mu}^t)\, C^{-1}\, (\mathbf{x}-\boldsymbol{\mu})$ is distributed as χ^2 with n degrees of freedom.

Confidence intervals from ML

- So, given a significance level α (or confidence level 1-α) and the # of parameters (= # of degrees of freedom to use), we can calculate the **maximum** value of X such that the probability that a chi-squared distributed variable will be $>X$ is α
- Since $-2\ ln\ L$ equals a number distributed as chi-squared + K, all values of **t** where $(-2 \ln L - \max(-2 \ln L)) < X$ will lie within the 1-α significance region.
  - For n=1, (1σ, 2σ, 3σ) limits on $\Delta(-2 \ln L)$ turn out to be 1, 4, 9
  - Use `stats.chi2.ppf(1-alpha,n)` to get the appropriate cutoff value for arbitrary n

Cosmological parameter determination: likelihoods from real data



Combining likelihoods from independent experiments

- If we combine the results of two independent experiments, the likelihood will be equal to the product of the likelihood from each experiment, so the **log likelihoods add**. We can then compare to the same chi-squared thresholds as for 1 experiment, as the **number of parameters** did not change.
- Let's look at a case where we have two orthogonal constraints:

```
fish1=np.array([[1,-1],[-1,1]])  # fisher matrix 1
fish2=np.array([[1,1],[1,1]])    # fisher matrix 2
nxy=101
xv= np.linspace(-5,5,nxy)
yv=xv
x,y = np.meshgrid(xv,yv)
```

Combining likelihoods from independent experiments

Examine the x and y arrays via bokeh and evaluate whether they contain what you expect: explore the values at different pixels.

```
# put plots in the notebook. We only need to do this once per notebook.
output_notebook()

# set up tooltips so we can read off values at the cursor
# We only need to run this code again when we change axis ranges.
p = figure(tooltips=[("x", "$x"), ("y", "$y"), ("value", "@image")])
p.x_range.range_padding = p.y_range.range_padding = 0

# You could instead try the EqHistColorMapper to map colors
color_mapper = LinearColorMapper(palette="Turbo256", low=x.min(), high=x.max())

# set up the image display, with axis ranges from -5 to +5
p.image(image=[y], x=-5, y=-5, dw=10, dh=10, level="image", \
    color_mapper=color_mapper)

# show the image, interactively
show(p)
```

Combining likelihoods from independent experiments

- Recall that (for large *n*) the likelihood can be described as a multivariate Gaussian, $\propto \exp(-1/2\ (\mathbf{x}^t - \mu^t)\, C^{-1}\, (\mathbf{x} - \mu)\ )$. Let's take $\mu = 0$, so we can write the log-likelihoods as:

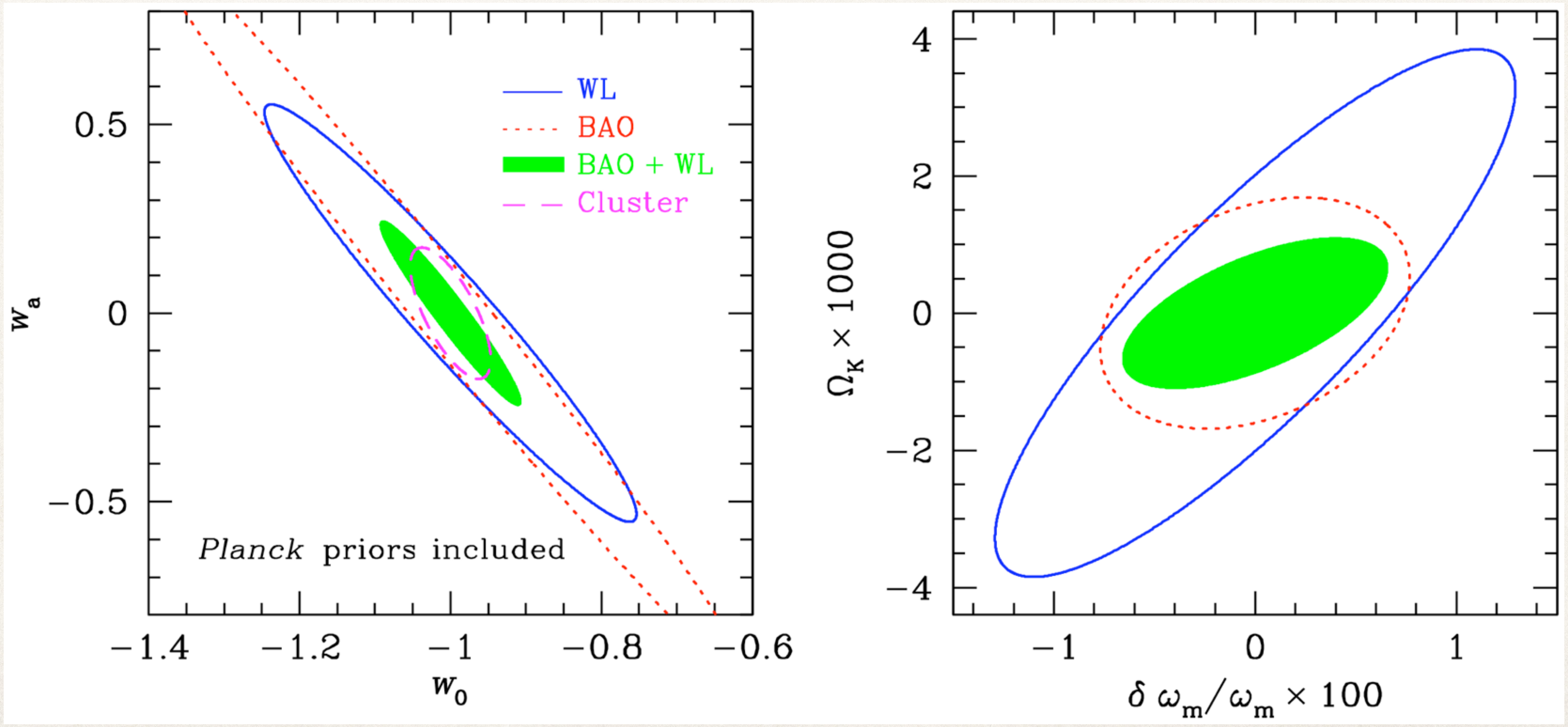
```
ll1=-.5*(fish1[0,0]*x**2+fish1[1,0]*x*y+fish1[0,1]*x*y+fish1[1,1]*y**2)
ll2=-.5*(fish2[0,0]*x**2+fish2[1,0]*x*y+fish2[0,1]*x*y+fish2[1,1]*y**2)
```

View the two log-likelihood arrays in bokeh (I have set up one for you)

```
# now we compare -2 ln L to cutoffs from chi-squared
cutoff=stats.chi2.ppf(1-signif,2)
is_ok = (-2*ll1 < cutoff)*1.
p.image(image=[is_ok], x=-5, y=-5, dw=10, dh=10, level="image",\
    color_mapper=color_mapper)
```

Also explore the shape of the region where -2 log-likelihood is below the cutoff for dataset 2 and for the combined dataset after you've looked at this for dataset 1...

Cosmological parameter determination: Fisher matrix predictions



Recipe for Maximum Likelihood

1. Write down $L=p(data \mid \boldsymbol{\theta})$, where $\boldsymbol{\theta}=[\theta_1, \theta_2,\ldots]$; let $LL = \ln L$. Then $LL = \ln p(data \mid \boldsymbol{\theta})$
2. Either:
   A. Figure out the analytic derivatives, $\delta LL / \delta \theta_i$. Find the point(s) where all of these derivatives = 0, which indicates that they must be minima, maxima, or saddle points.
   B. Write a Python function that returns $-LL$ for your actual dataset, given the value of the parameters. You can then use a variety of Python routines to find the minimum of that function (most can be applied via `scipy.optimize.minimize`)
   C. Set up a grid of possible values for $\boldsymbol{\theta}$. Calculate LL at each point of the grid; choose the maximum value; or
   D. apply Markov Chain Monte Carlo methods to explore $p(data \mid \boldsymbol{\theta})$
3. Check that your solution is actually a maximum of LL.

Applying to a previous problem

- In part 2 of Homework 3, we tested various statistics for measuring the Gaussian velocity dispersion (sigma) of galaxies in a cluster, in the presence of a background with a flat redshift distribution. ALL of them were biased, in one direction or another, even with 100 galaxies in the cluster.
  - Can we do better with ML?
- Let's set up a test case: 10^3 galaxies, of which a fraction $f_{outlier}$ have redshifts evenly distributed between 0 and 6500 km/sec, and the rest follow $N(3150,930^2)$. Let's assume we know the mean z and want to measure the σ and $f_{outlier}$.

Setting up our mock data

```
import numpy.random as random
ndata=int(1E3)
foutlier=0.1
z0=3150.

isoutlier=random.rand(ndata) < foutlier

zdist=(1-isoutlier)*(random.randn(ndata)*930.+z0) +
(isoutlier)*random.rand(ndata)*6500

plt.hist(zdist,bins=100)
```

Recipe for ML

1. Write down $L=p(data \mid \boldsymbol{\theta})$, where $\boldsymbol{\theta}=[\theta_1, \theta_2,\ldots]$; let $LL = \ln L$. Then $LL = \ln p(data \mid \boldsymbol{\theta})$

- Given `z0` = true mean z, `zdist`=observed z distribution, `sigmas`= σ parameter value, `foutliers` = $f_{outlier}$ parameter value, the calculations would look like:

```
p = np.product( (1-foutliers)*1/np.sqrt(2.*np.pi*sigmas**2) *np.exp(-(zdist-
z0)**2/2/sigmas**2 ) + foutliers*(1./6500.))
```

- However, it is more numerically stable to work with log likelihood:

```
LL = np.sum( np.log((1-foutliers)*1/np.sqrt(2.*np.pi*sigmas**2) *np.exp(-(zdist-
z0)**2/2/sigmas**2 ) + foutliers*(1./6500.)))
```