

p(y): Python for Scientific Inference



Berian James

Overview of lecture material

- Symbolic computation in Python
sympy, sage
- Introduction to (Bayesian) inference
- Bayesian inference in Python
emcee, pymc

I. Symbolic computation in Python

Symbolic mathematics with Python

- <http://sympy.org/>
SymPy home page
- <http://docs.sympy.org>
Reference, tutorial
- Perhaps think of SymPy as Mathematica for Python, including integration, geometry, linear algebra, statistics, ODE solving and tensor algebra

```
>>> import sympy
```

- <https://github.com/sympy/sympy/wiki>
SymPy wiki

Symbolic computation

- See the wiki for comparison with other symbolic computation systems, e.g.:
<https://github.com/sympy/sympy/wiki/SymPy-vs.-Mathematica>
<https://github.com/sympy/sympy/wiki/SymPy-vs.-Maple>
<https://github.com/sympy/sympy/wiki/SymPy-vs.-Matlab>
- Computer algebra systems have existed since the 1960s, developed primarily within the theoretical physics and artificial intelligence communities until the appearance of Maple and Mathematica in the 1980s. Many problems in symbolic computation remain unsolved; many problems have been solved theoretically, but not implemented.
- arXiv categories: cs.SC (symbolic computation), cs.MS (mathematical software).

Sympy architecture: classes and objects

- Abstract objects are represented with the Basic class, of which Symbol, Function, ... are subclasses. All variables which will be assigned such an object must be declared as such in advance.

Sympy architecture: classes and objects

- Abstract objects are represented with the Basic class, of which Symbol, Function, ... are subclasses. All variables which will be assigned such an object must be declared as such in advance.
- `isympy` preloads the package and assigns a handful of Symbols.

```
~ > isympy
```

```
IPython console for SymPy 0.7.1 (Python 2.7.2-32-bit)  
(ground types: python)
```

These commands were executed:

```
>>> from __future__ import division  
>>> from sympy import *  
>>> x, y, z, t = symbols('x y z t')  
>>> k, m, n = symbols('k m n', integer=True)  
>>> f, g, h = symbols('f g h', cls=Function)
```

Sympy modules

- The symbolic computation framework for Sympy is provided by `sympy.core`. Everything else is in modules for performing various mathematical tasks.
- See <http://docs.sympy.org/dev/modules/>

Integration: the Risch-Norman algorithm

- The Risch algorithm checks whether a given expression has an antiderivative and, as a by-product, returns the antiderivative if it does.
- A restricted (but fast) implementation, the Risch-Norman algorithm, exists within SymPy. (Axiom is the only system that implements Risch completely, Mathematica has a number of proprietary developments too.)

Integration: the Risch-Norman algorithm

- The Risch algorithm checks whether a given expression has an antiderivative and, as a by-product, returns the antiderivative if it does.
- A restricted (but fast) implementation, the Risch-Norman algorithm, exists within SymPy. (Axiom is the only system that implements Risch completely, Mathematica has a number of proprietary developments too.)

```
>>> integrate(x/(x**2+2*x+1), x)
```

Integration: the Risch-Norman algorithm

- The Risch algorithm checks whether a given expression has an antiderivative and, as a by-product, returns the antiderivative if it does.
- A restricted (but fast) implementation, the Risch-Norman algorithm, exists within SymPy. (Axiom is the only system that implements Risch completely, Mathematica has a number of proprietary developments too.)

```
>>> integrate(x/(x**2+2*x+1), x)
```

```
1
log(x + 1) + ----
x + 1
```

Linear algebra and matrices

```
>>> M = Matrix( ([1,2,3],[4,5,6],[7,8,9]) )
```

Linear algebra and matrices

```
>>> M = Matrix( ([1,2,3],[4,5,6],[7,8,9]) )  
>>> M * M
```

Linear algebra and matrices

```
>>> M = Matrix( ([1,2,3],[4,5,6],[7,8,9]) )
>>> M * M
[ 30    36    42 ]
[      ]
[ 66    81    96 ]
[      ]
[ 102   126   150 ]
```

Linear algebra and matrices

```
>>> M = Matrix([1,2,3],[4,5,6],[7,8,9])
```

```
>>> M * M
```

```
[30    36    42 ]
```

```
[
```

```
[66    81    96 ]
```

```
[
```

```
[102   126   150]
```

```
>>> x,y = symbols('x,y')
```

```
>>> M = Matrix([[x,y],[y,x]])
```

```
>>> M.det()
```

Linear algebra and matrices

```
>>> M = Matrix([1,2,3],[4,5,6],[7,8,9])
```

```
>>> M * M
```

```
[30  36  42 ]  
[      ]  
[66  81  96 ]  
[      ]  
[102 126 150]
```

```
>>> x,y = symbols('x,y')
```

```
>>> M = Matrix([[x,y],[y,x]])
```

```
>>> M.det()
```

```
  2      2  
x  - y
```


Statistical computation: sympy.statistics

- Aims to provide abstract structures that represent probability distributions. Currently only two pre-defined classes (univariate normal and uniform), but the underlying API is fairly extensible.

```
>>> N = Normal(1, 1)
```

Statistical computation: sympy.statistics

- Aims to provide abstract structures that represent probability distributions. Currently only two pre-defined classes (univariate normal and uniform), but the underlying API is fairly extensible.

```
>>> N = Normal(1, 1)
```

```
>>> N.pdf(1)
```

Statistical computation: sympy.statistics

- Aims to provide abstract structures that represent probability distributions. Currently only two pre-defined classes (univariate normal and uniform), but the underlying API is fairly extensible.

```
>>> N = Normal(1, 1)
```

```
>>> N.pdf(1)
```

$$\frac{1}{\sqrt{2}}$$

$$2 * \sqrt{\pi}$$

Statistical computation: sympy.statistics

- Aims to provide abstract structures that represent probability distributions. Currently only two pre-defined classes (univariate normal and uniform), but the underlying API is fairly extensible.

```
>>> N = Normal(1, 1)
```

```
>>> N.pdf(1)
```

$$\frac{1}{\sqrt{2}}$$

$$2 * \sqrt{\pi}$$

```
>>> N.pdf(3).evalf()
```

Statistical computation: sympy.statistics

- Aims to provide abstract structures that represent probability distributions. Currently only two pre-defined classes (univariate normal and uniform), but the underlying API is fairly extensible.

```
>>> N = Normal(1, 1)
```

```
>>> N.pdf(1)
```

$$\frac{1}{\sqrt{2}}$$

$$2 * \sqrt{\pi}$$

```
>>> N.pdf(3).evalf()
```

```
0.0539909665131881
```

Statistical computation: sympy.statistics

- Symbolic use also available, of course.

Statistical computation: sympy.statistics

- Symbolic use also available, of course.

```
>>> x = Symbol('x')
```

```
>>> N.cdf(x)
```

Statistical computation: sympy.statistics

- Symbolic use also available, of course.

```
>>> x = Symbol('x')
```

```
>>> N.cdf(x)
```

$$\frac{\operatorname{erf}\left(\frac{\sqrt{2} * (x - 1)}{\sqrt{2}}\right)}{2} + \frac{1}{2}$$

Statistical computation: sympy.statistics

- Symbolic use also available, of course.

```
>>> x = Symbol('x')
```

```
>>> N.cdf(x)
```

$$\text{erf}\left(\frac{\sqrt{2} * (x - 1)}{2}\right) + \frac{1}{2}$$

```
>>> N.cdf(-oo), N.cdf(1), N.cdf(oo)
```

Statistical computation: sympy.statistics

- Symbolic use also available, of course.

```
>>> x = Symbol('x')
```

```
>>> N.cdf(x)
```

$$\text{erf}\left(\frac{\sqrt{2} * (x - 1)}{2}\right) + \frac{1}{2}$$

```
>>> N.cdf(-oo), N.cdf(1), N.cdf(oo)
(0, 1/2, 1)
```

Statistical computation: sympy.statistics

- Symbolic use also available, of course.

```
>>> x = Symbol('x')
```

```
>>> N.cdf(x)
```

$$\text{erf}\left(\frac{\sqrt{2} * (x - 1)}{2}\right) + \frac{1}{2}$$

```
>>> N.cdf(-oo), N.cdf(1), N.cdf(oo)
```

```
(0, 1/2, 1)
```

```
>>> N.cdf(5).evalf()
```

Statistical computation: sympy.statistics

- Symbolic use also available, of course.

```
>>> x = Symbol('x')
```

```
>>> N.cdf(x)
```

$$\text{erf}\left(\frac{\sqrt{2} * (x - 1)}{2}\right) + \frac{1}{2}$$

```
>>> N.cdf(-oo), N.cdf(1), N.cdf(oo)
```

```
(0, 1/2, 1)
```

```
>>> N.cdf(5).evalf()
```

```
0.999968328758167
```

Sympy: output and rendering

- In the terminal, unicode rendering is available via the `pprint` function.
- Output to LaTeX markup is available via the `latex` function:

```
>>> latex(Integral(x**2, x))  
\int x^{2}\,, dx
```
- Python printing is helpful for generating sympy commands quickly:

```
>>> print python(Integral(x**2, x))  
x = Symbol('x')  
e = Integral(x**2, x)
```
- From the iPython notebook, the `%load_ext sympyprinting` magic will produce rendered LaTeX output.

Symbolic computation breakout (10 - 15 mins)

- Implement a bivariate normal distribution in Sympy
http://en.wikipedia.org/wiki/Multivariate_normal_distribution
- There are many ways to do this!
- The `.subs()` method may be useful.

II. Overview of Bayesian inference

Approaches to statistical inference

- For scientists, the rôle of inference is to draw quantitative conclusions from noisy data.
- Historically, approaches to inference can be divided into two* camps termed ‘frequentist’ and ‘Bayesian’. The frequentist interpretation of probability is expressed in terms of repeated trials, while Bayesian interpret probability as a degree of belief.
- Reading material: Blog post from yesterday (11/18/12) by Larry Wasserman <http://normaldeviate.wordpress.com/2012/11/17/what-is-bayesianfrequentist-inference/>
- Reading material: Poincaré’s intervention in the Dreyfus trial (ca. 1900) <http://www.maths.ed.ac.uk/~aar/dreyfus.htm>

Approaches to statistical inference

- The methodological implications of this distinction are profound and subtle. The notion of hypothesis testing is drawn from frequentist statistics, where propositions are evaluated by the possibility of their being false (e.g. p-values). On the other hands, concepts such as likelihood and evidence arise from within Bayesian statistics.
- While the distinction is not considered a major rift within statistics today, the application of inference within scientific fields is often surprisingly one-sided: e.g., within cosmology Bayesian inference is standard, in particle physics frequentist statistics are the norm. Astronomy as a whole is more mixed, what about other fields?

Approaches to statistical inference in science

- Ultimately, the difference between frequentist and Bayesian statistics is not of the highest practical importance for scientists. The kinds of questions that matter to scientists are:
 - “I have these data with some error bars (that, between the two of us, I do not trust). I want to publish in Nature. What do I do in between?”
 - Or, more specifically: “How do I fit a model to these data, or decide which of two models is better?”
 - Or, even more specifically: “How do I take this numpy array and find maximum likelihood parameters with an associated covariance matrix and/or joint probability distributions (while leaving enough time to finish my AY250 final project)?”

Approaches to statistical inference **with Python**

- The goal of this lecture is to provide you with tools for inference with real data in Python.
 - “How do I take this numpy array and find maximum likelihood parameters with an associated covariance matrix and/or joint probability distributions while leaving enough time to finish my AY250 final project?”

Approaches to statistical inference **with Python**

- The goal of this lecture is to provide you with tools for inference with real data in Python.

- “How do I take this numpy array and find maximum likelihood parameters with an associated covariance matrix and/or joint probability distributions while leaving enough time to finish my AY250 final project?”

Packages that interface with numpy,

Approaches to statistical inference **with Python**

- The goal of this lecture is to provide you with tools for inference with real data in Python.

- “How do I take this **numpy array** and **find maximum likelihood parameters** with an associated covariance matrix and/or joint probability distributions while leaving enough time to finish my AY250 final project?”

Packages that interface with numpy,
return ‘optimised’ numbers (that are probably
arguments in a function call),

Approaches to statistical inference **with Python**

- The goal of this lecture is to provide you with tools for inference with real data in Python.

- “How do I take this numpy array and find maximum likelihood parameters with an associated covariance matrix and/or joint probability distributions while leaving enough time to finish my AY250 final project?”

Packages that interface with numpy, return ‘optimized’ numbers (that are probably arguments in a function call), as well as some description of the probability distribution from which they are drawn (an array? a function to draw samples?)

Review of material from earlier in the course

- <http://scikit-learn.org>
scikit-learn homepage, machine learning in Python
- <http://scikits.appspot.com/statsmodels>
statsmodels homepage, download, installation
- <http://statsmodels.sourceforge.net/>
statsmodels documentation, API reference, examples; not complete

Bayesian parameter inference: formalism

- When embarking upon an experiment, we almost always have some prior expectation about the outcome. Bayesian inference is the process by which this expectation is updated to account for new data we obtain.
- Information about parameters is expressed in terms of probability distributions:

$$\underbrace{p(\boldsymbol{\theta}|D, I)}_{\text{'posterior'}} \propto \underbrace{p(D|\boldsymbol{\theta}, I)}_{\text{'likelihood'}} \underbrace{p(\boldsymbol{\theta}|I)}_{\text{'prior'}}$$

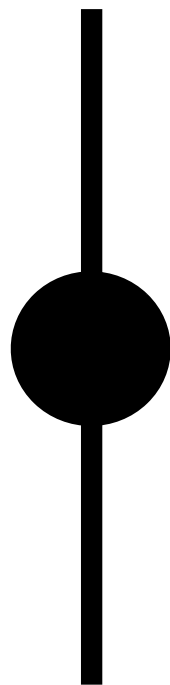
Evaluating the likelihood

- Much of the work in inference is in generating the likelihood

$$\mathcal{L}(\boldsymbol{\theta}) = [\mathcal{L}(\boldsymbol{\theta}|D)] = p(D|\boldsymbol{\theta})$$

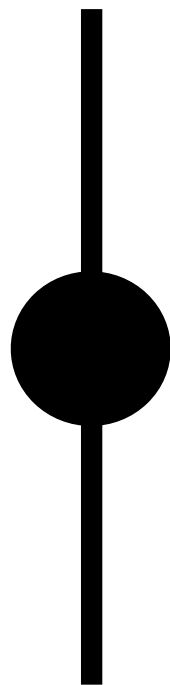
Data points are secretly probability distributions

Top view

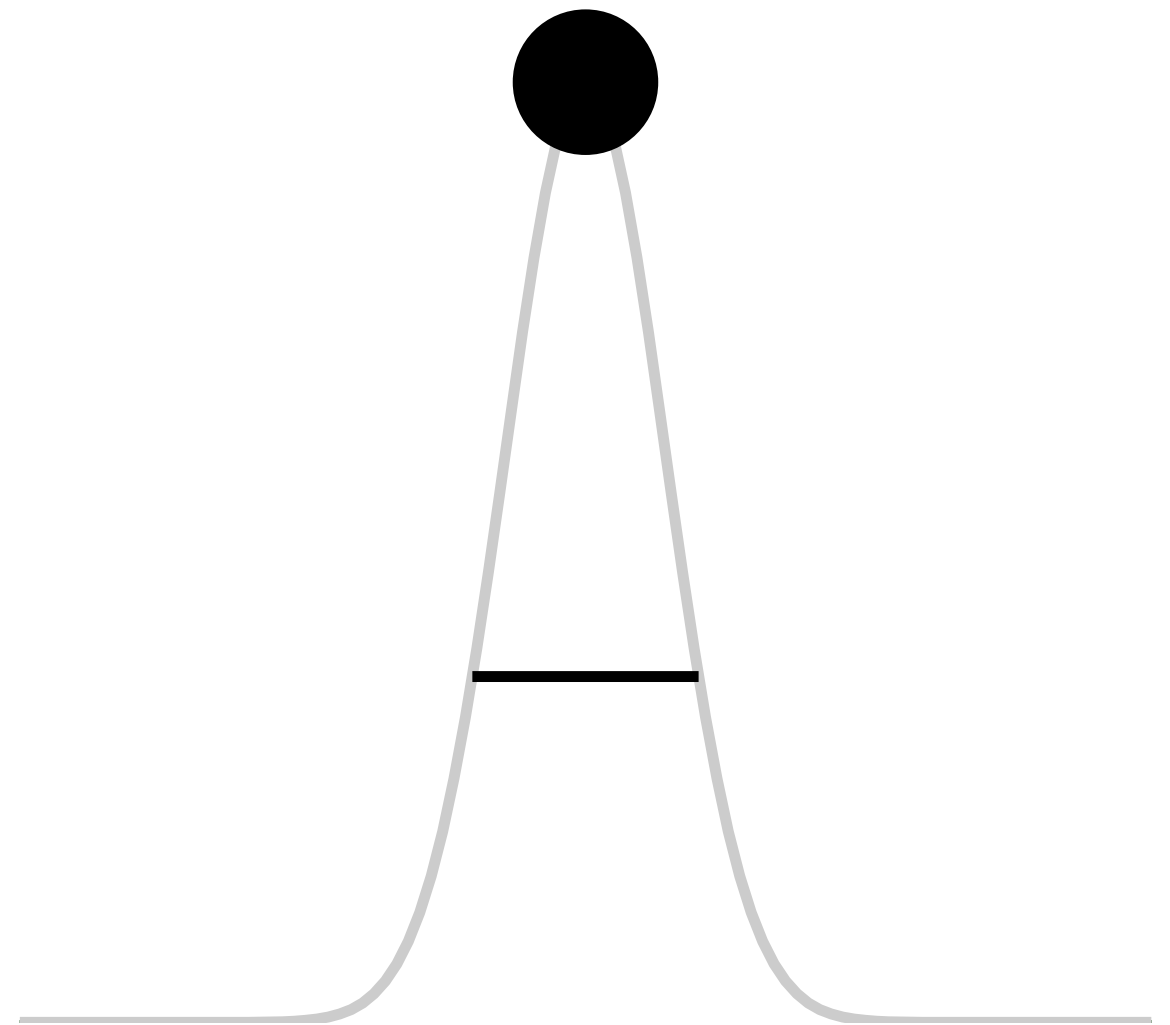


Data points are secretly probability distributions

Top view

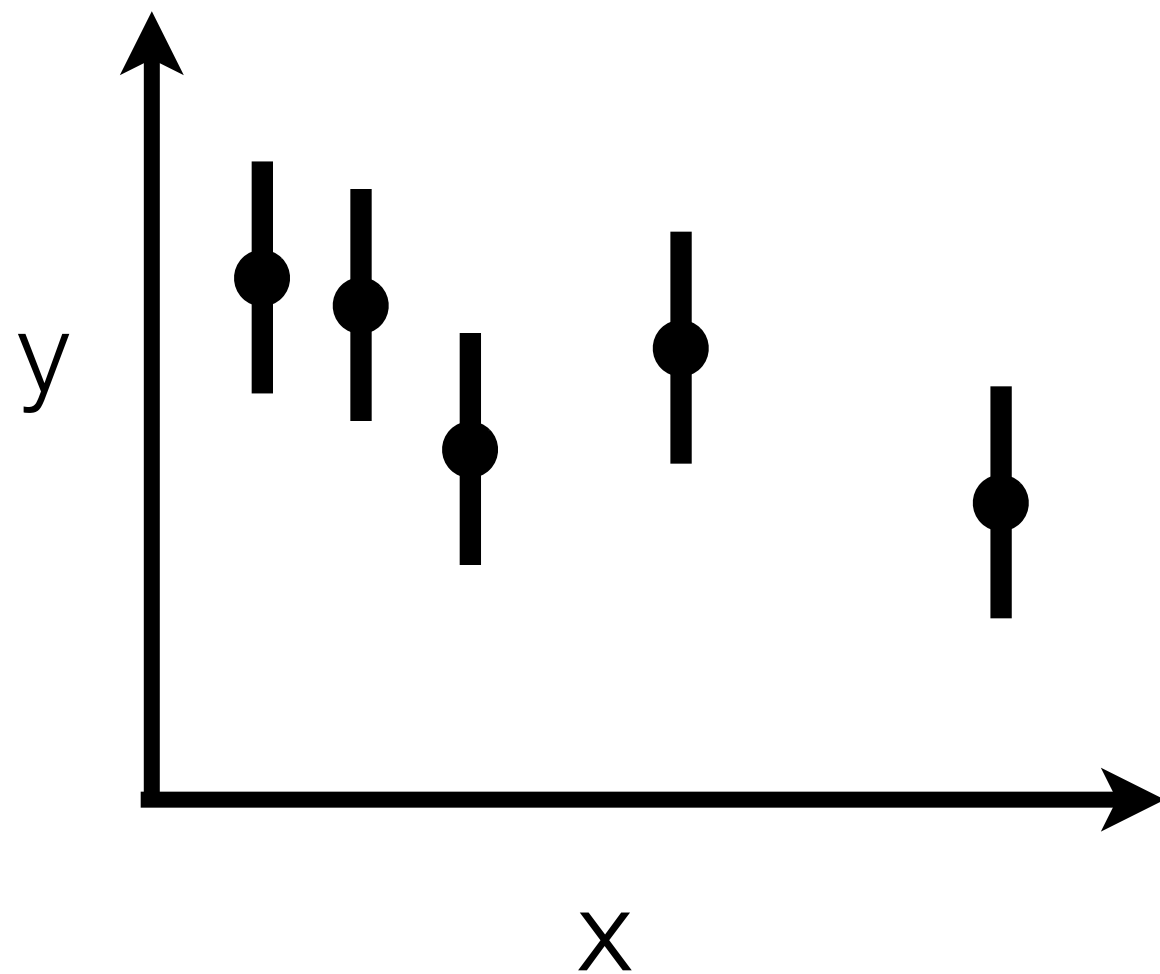


Side view

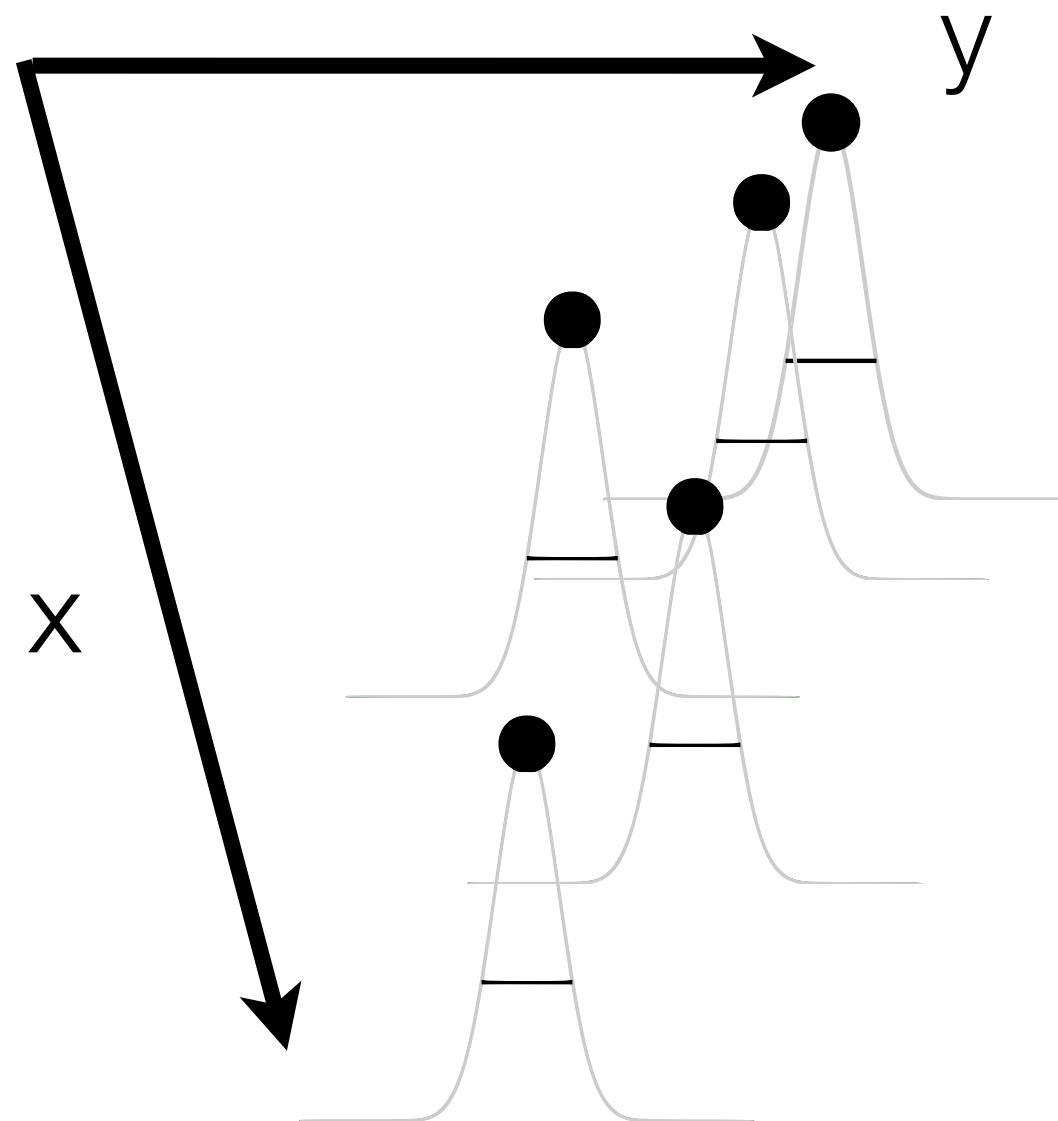


Data points are secretly probability distributions

Top view



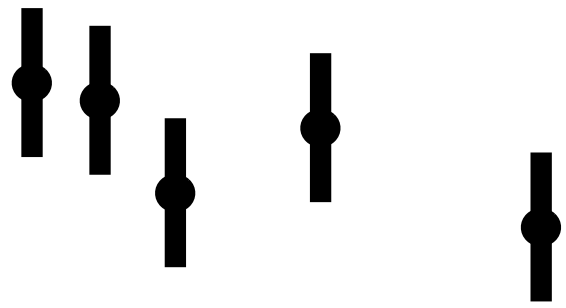
Side view



Evaluating the likelihood

- Much of the work in inference is in generating the likelihood

$$\mathcal{L}(\boldsymbol{\theta}) = [\mathcal{L}(\boldsymbol{\theta}|D)] = p(D|\boldsymbol{\theta})$$


$$\Rightarrow \mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^n p_i(D_i|\boldsymbol{\theta})$$

$$\log \mathcal{L} = \sum_{i=1}^n \log p_i(D_i|\boldsymbol{\theta})$$

Inference with normal distributions

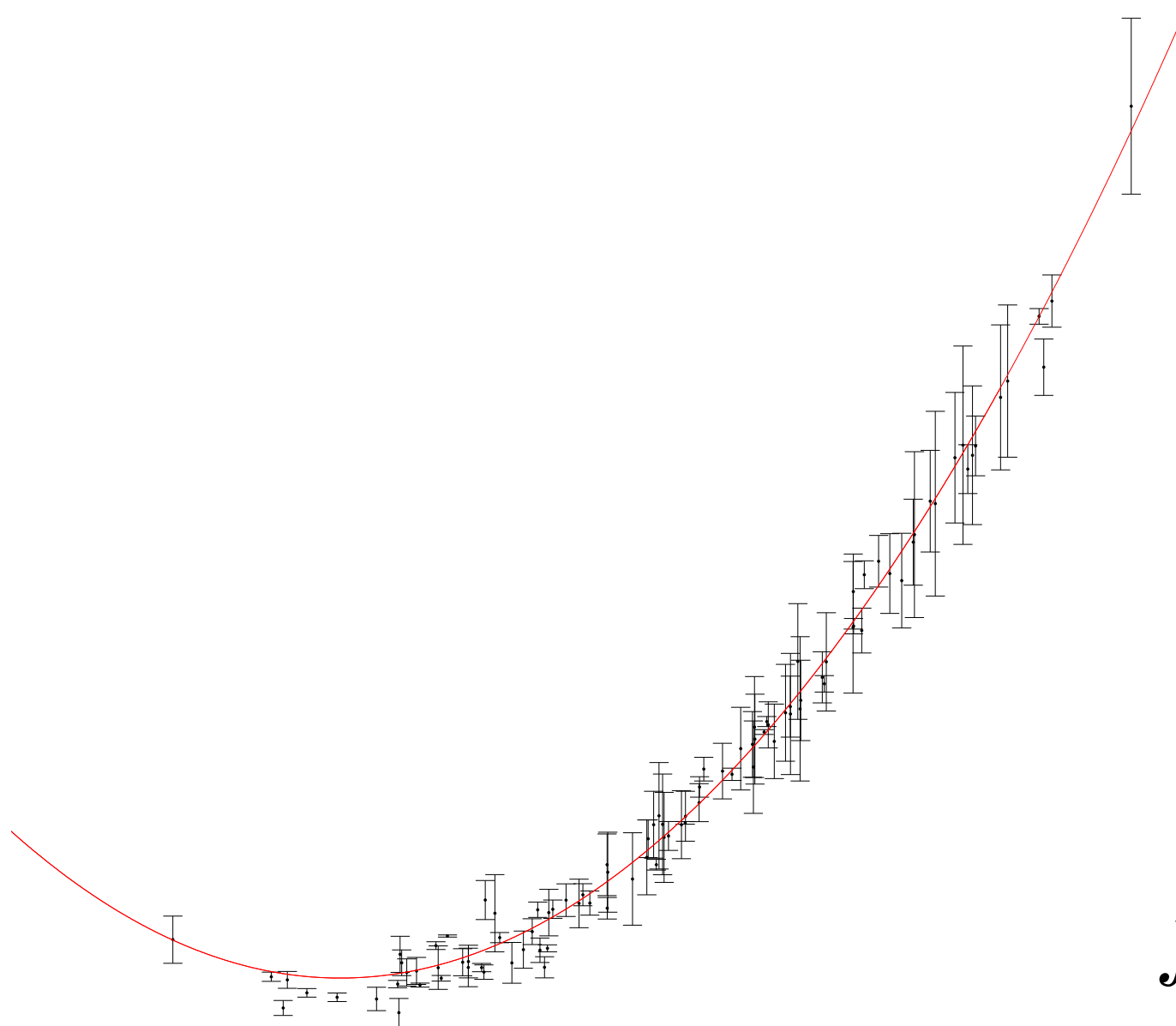
- Normal distributions (a data set with uncorrelated normally distributed uncertainty) is analytically tractable, and very commonly used in scientific inference.

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^n \exp \left[-\frac{1}{2} \left(\frac{y_i - f(x_i|\boldsymbol{\theta})}{\sigma_i} \right)^2 \right]$$

$$\Rightarrow -2 \log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^n \left(\frac{y_i - f(x_i|\boldsymbol{\theta})}{\sigma_i} \right)^2$$

Example: Evaluating a likelihood

- Uncorrelated data set



$$f(x) = a_1 x^2 + a_2$$

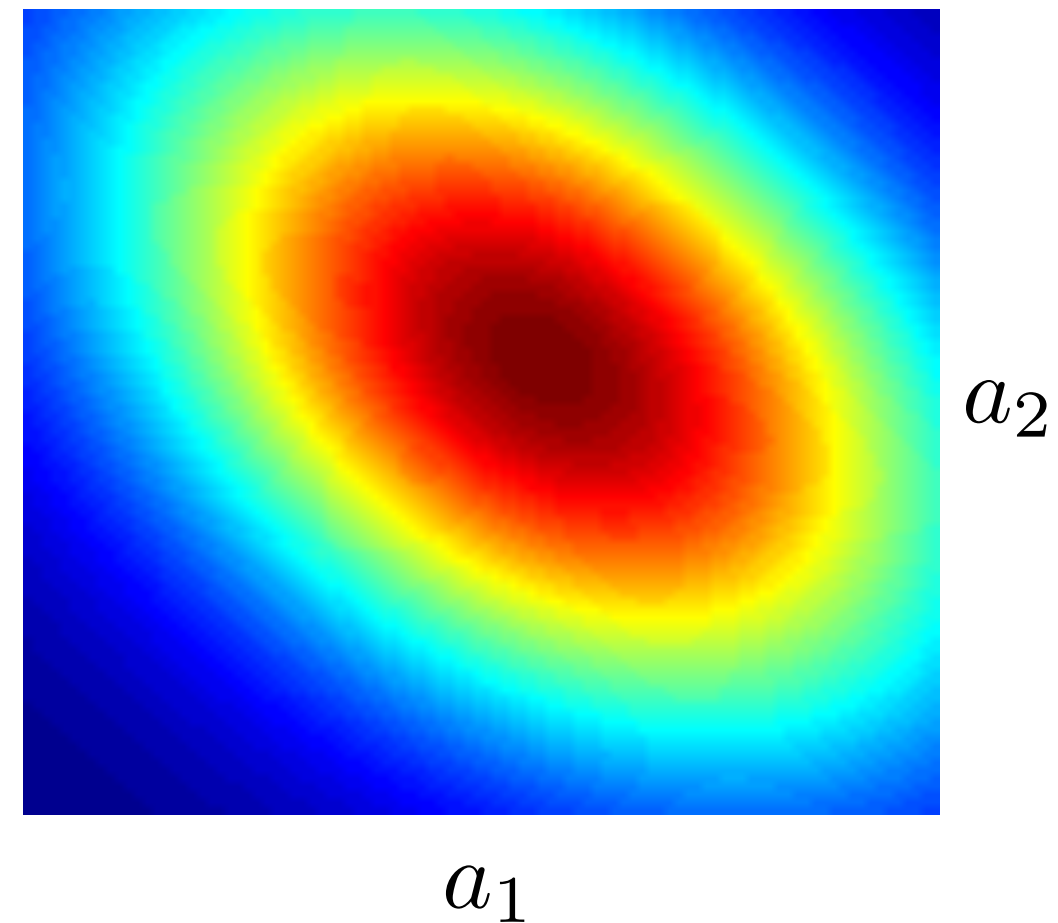
Example: Evaluating a likelihood

- Likelihood surface for grid of parameter values (a_1, a_2)

$$f(x|a_1, a_2) = a_1 x^2 + a_2$$

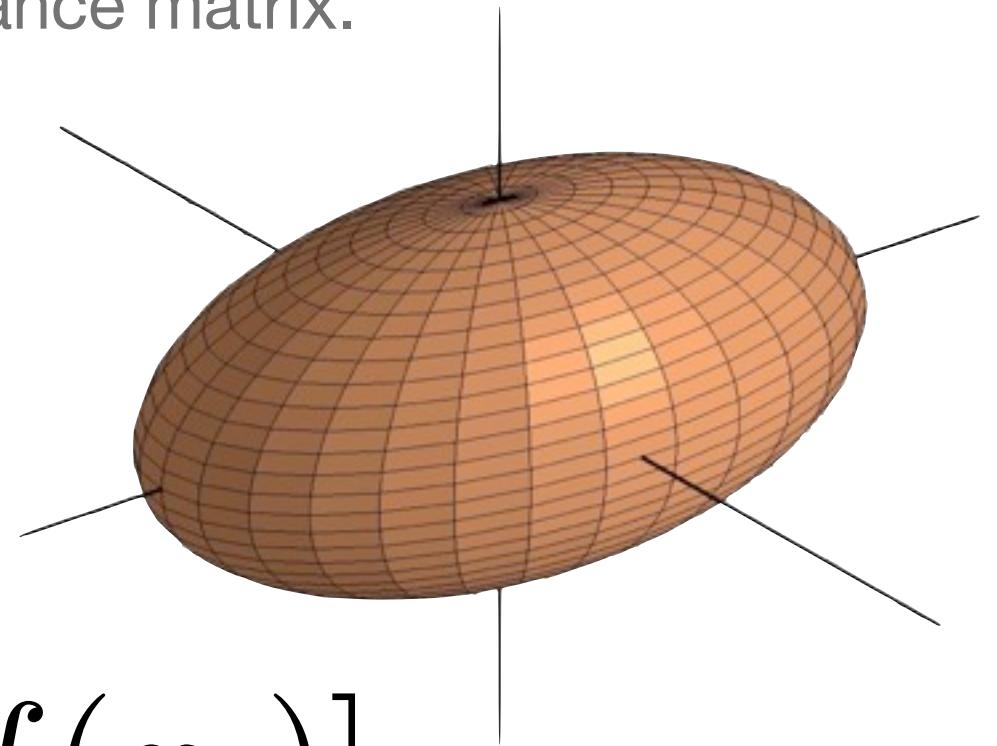
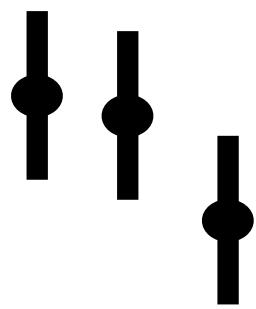
$$-2 \log \mathcal{L} = \sum \left(\frac{y_i - f(x_i|a_1, a_2)}{\sigma_i} \right)^2$$

$$\mathcal{L}(a_1, a_2) = \exp(-\log \mathcal{L}/2)$$



Inference with correlated data points

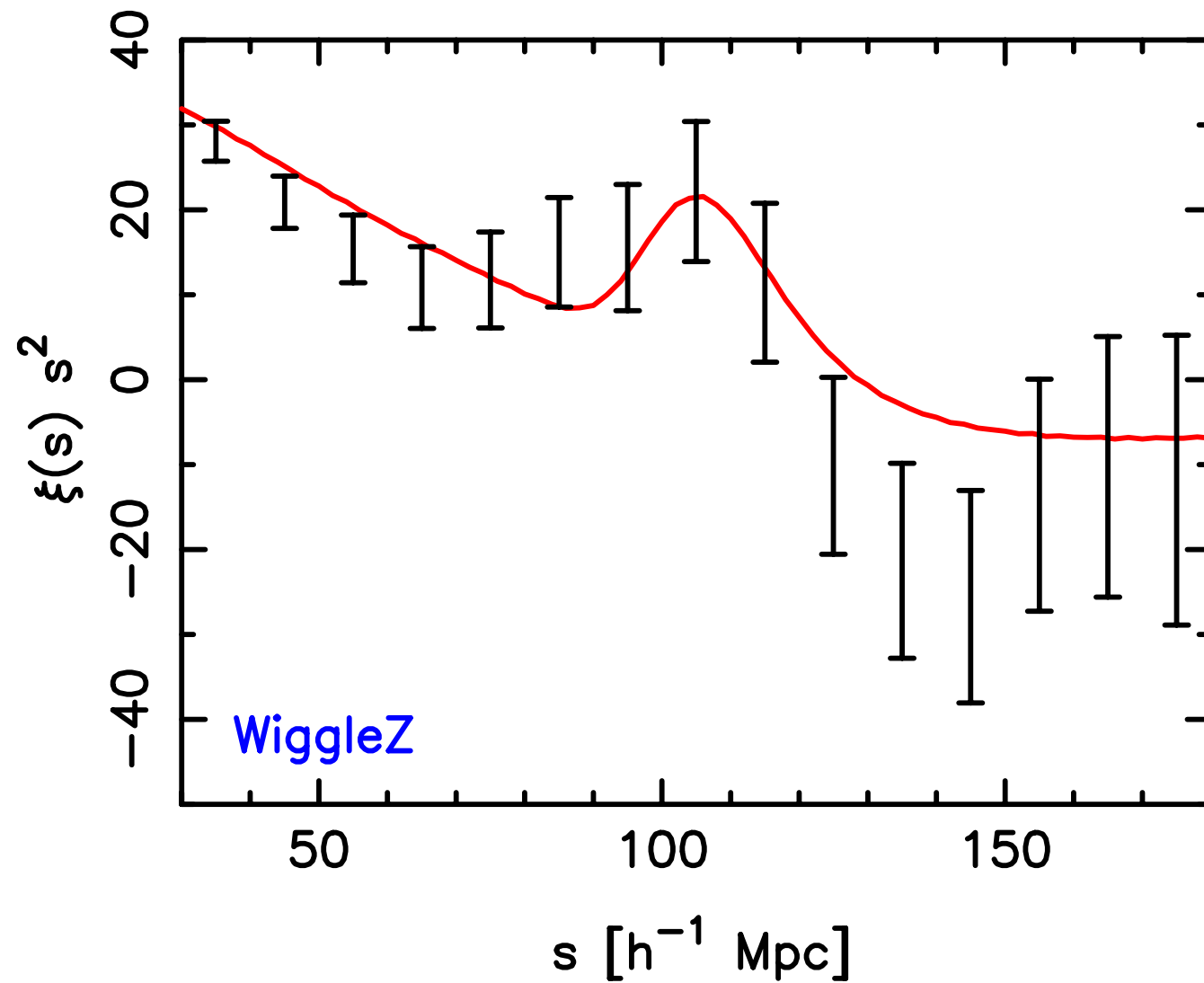
- When data points are correlated, the data set as a whole is represented as a multivariate normal distribution with a covariance matrix.



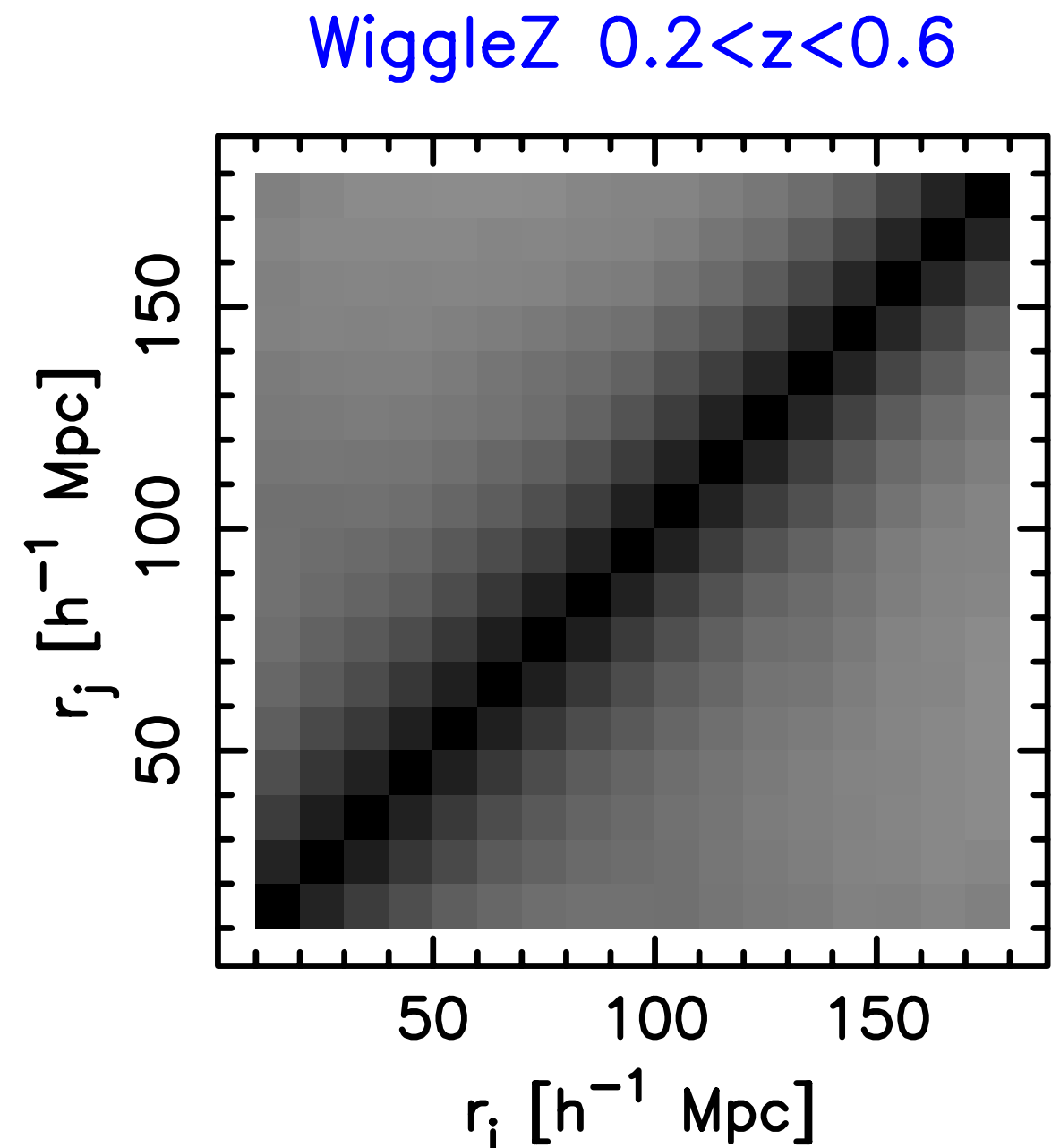
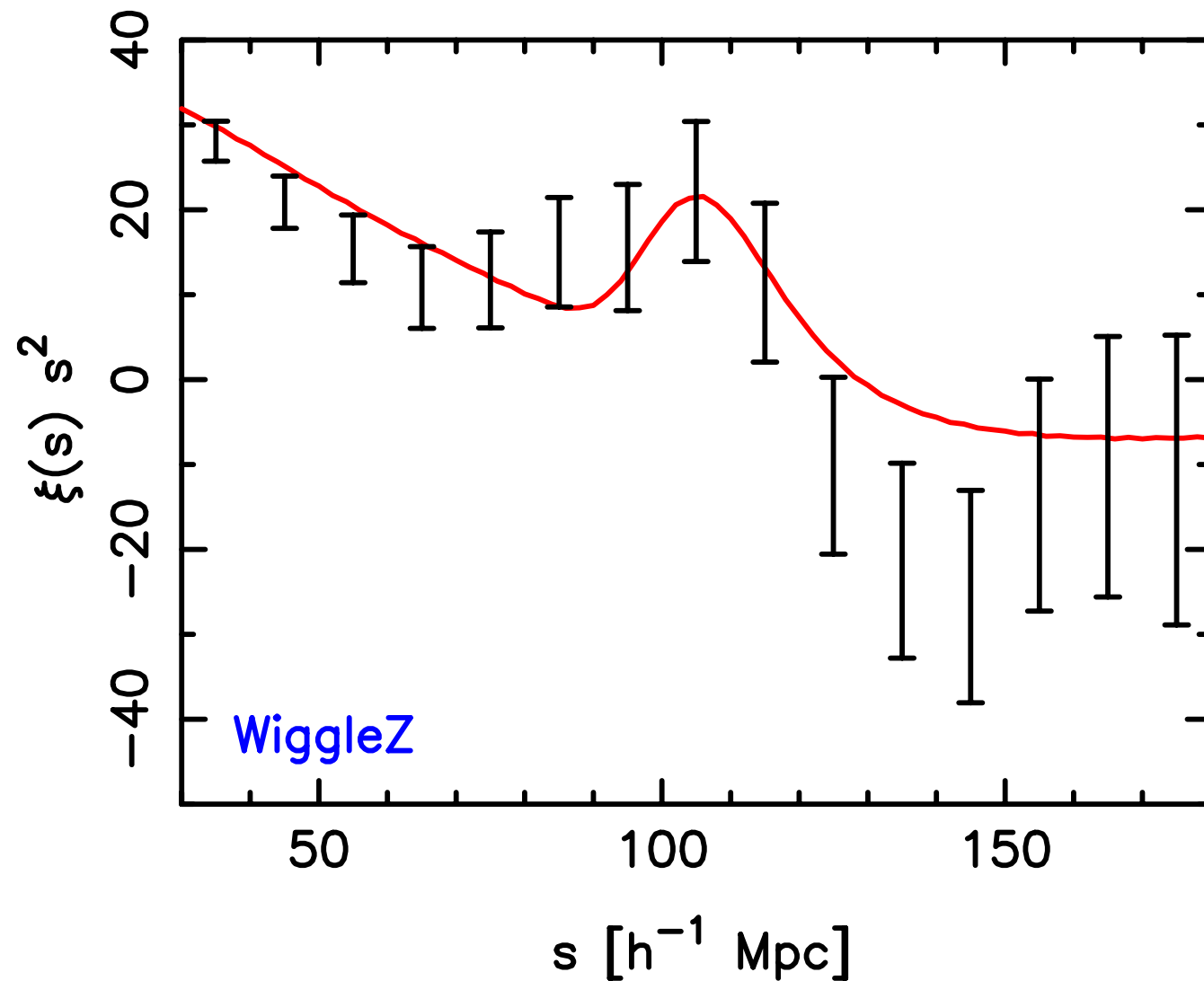
$$\Delta = [y_i - f(x_i)]$$

$$-2 \log \mathcal{L}(\boldsymbol{\theta}) = \Delta \mathbf{C}^{-1} \Delta^T$$

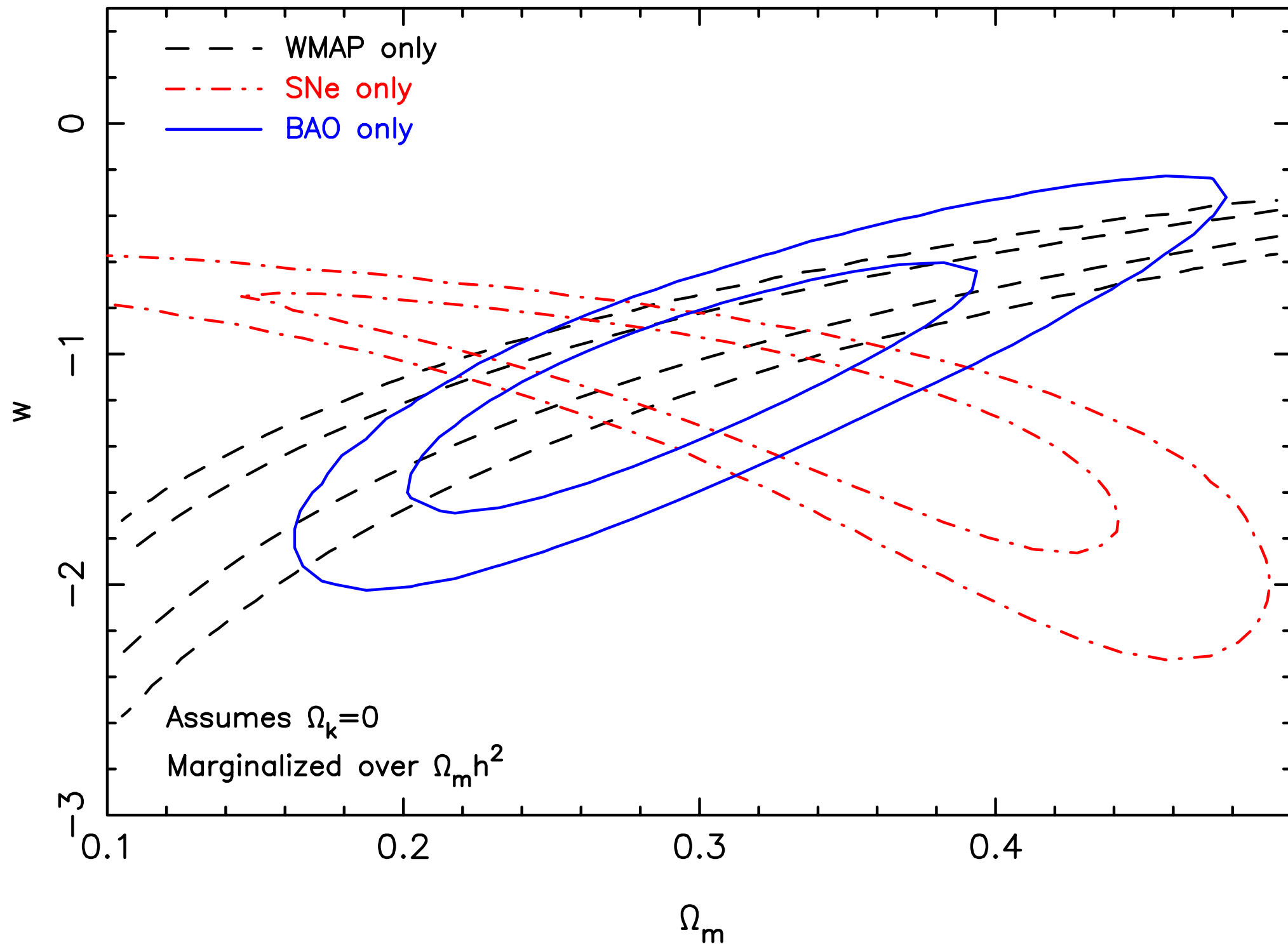
Example: Inference with correlated data points



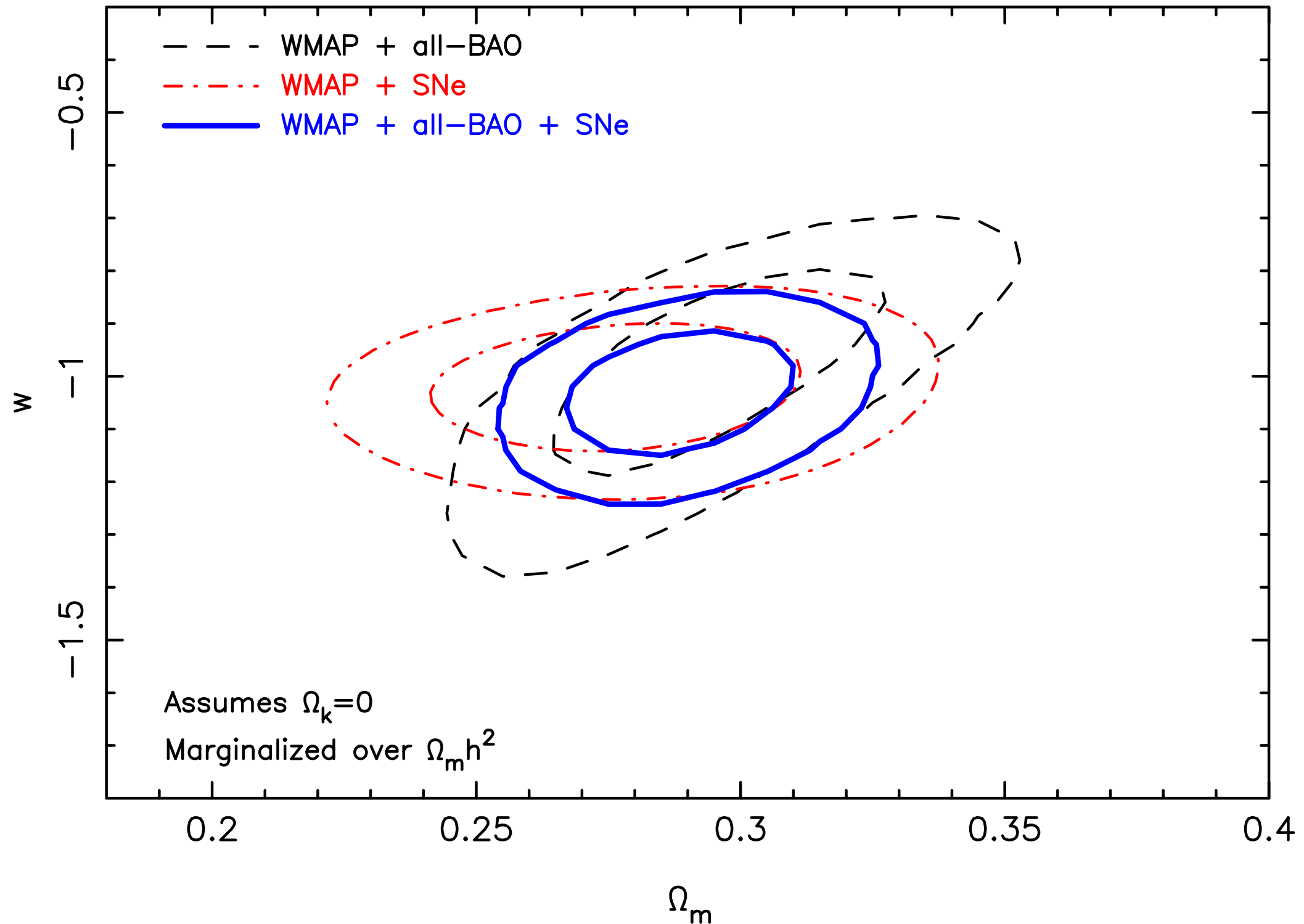
Example: Inference with correlated data points



Example: Inference with correlated data points



Example: Inference with correlated data points



Higher-dimensional parameter spaces

- A grid search of parameter values becomes slow for $n > 3$, depending on how patient you are. Optimization routines (fmin, fmin_bfgs, etc) can be used to locate the maximum likelihood (or maximum a posteriori) region of parameter space.
- If the parameter distribution is multivariate normal, the covariance matrix can be evaluated analytically or numerically. For many scientific experiments, this can be sufficient for inference.
- But not always. Non-gaussian parameter distributions require other methods, and that is where Monte-Carlo Markov Chain routines can help.

Bayesian inference break-out (20 - 30 mins)

- Using the provided data set, write a model function of the form $f(x) = a \log(x) + b$ and another function to evaluate the likelihood given a parameter value. Assume $b = 0$ to begin with.
- For a range of parameter values, compute the likelihood. Find the maximum-likelihood parameter value. Characterize the distribution for this parameter value.
- Now include prior knowledge for the value of a . E.g., a previous experiment has told you that $p(a) \sim N(0.8, 0.3)$. Given this prior knowledge of the parameter distribution, update the prior to the posterior.
- Extension: Let b be a free parameter too, and plot the joint distribution of a and b . Compute the marginal distribution of a .