

# Estimation of hierarchical open and closed-population models with MCMC using PyMC

Christopher J Fonnesbeck and Michael J Conroy



The University of Georgia

PyMC  
Python  
MCMC

# Python

Python Programming Language -- Official Website

http://python.org/ Ask.com

Atlanta Weather del.icio.us Backpack Flickr Gmail DivShare Trichech.us PyMC + del.icio.us Translate TinyURL

python™

ABOUT >

- Getting Started
- Applications
- Success Stories
- Quotes
- Website
- Help

NEWS >

DOCUMENTATION >

DOWNLOAD >

COMMUNITY >

FOUNDATION >

CORE DEVELOPMENT >

LINKS >

Quick Links (2.5)  
» Documentation  
» Windows Installer  
» Source Distribution  
» Package Index

Python Jobs

Donate to the PSF

Report website bug

## The Python Programming Language

Python® is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. Many Python programmers report substantial productivity gains and feel the language encourages the development of higher quality, more maintainable code.

Python runs on Windows, Linux/Unix, Mac OS X, OS/2, Amiga, Palm Handhelds, and Nokia mobile phones. Python has also been ported to the Java and .NET virtual machines.

Python is distributed under an OSI-approved open source license that makes it free to use, even for commercial products.

The Python Software Foundation (PSF) holds and protects the intellectual property rights behind Python, underwrites the PyCon conference, and funds grants and other projects in the Python community.

Read more -or- [try Python now](#)

» **PyCon 2007 Funding Available**  
The PSF is making financial aid available for attending [PyCon 2007](#). The deadline for applications is January 10. For details, see [PyCon 2007 Funding Available](#).

NASA uses Python...  
... so does Rackspace, Industrial Light and Magic, AstraZeneca, Honeywell, and many others.

What they are saying...

"Python is fast enough for our site and allows us to produce maintainable features in record times, with a minimum of developers."

--Cuong Do,  
[YouTube.com](#)

more...

Using Python For...

» Web Programming  
CGI, Zope, Django, TurboGears, XML  
» Databases

Powerful  
AND  
Easy to Use

Interpreted

# Object-oriented

# Dynamic Types

# Automatic Memory Management

# Extensible

Free and Open

# 3rd Party Modules

SciPy -

http://scipy.org/ Google

Atlanta Weather del.icio.us Backpack Flickr Gmail DivShare Trichech.us PyMC + del.icio.us Translate TinyURL Login

SciPy.org Sponsored By ENTHOUGHT Search Titles Text

**Wiki**

- SciPy
- Documentation
- Mailing Lists
- Download
- Installing SciPy
- Topical Software
- Cookbook
- Developer Zone
- RecentChanges
- FindPage

**Page**

- Immutable Page
- Info
- Attachments

More Actions:

**SciPy**

## Scientific Tools for Python

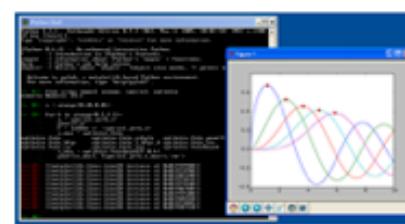
SciPy (pronounced "Sigh Pie") is open-source software for mathematics, science, and engineering. It is also the name of a very popular [conference on scientific programming with Python](#). The core library is NumPy which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays, and provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install, and are free of charge. NumPy and SciPy are easy to use, but powerful enough to be depended upon by some of the world's leading scientists and engineers. If you need to manipulate numbers on a computer and display or publish the results,

give SciPy a try!

## Python and Scientific Computing

NumPy and SciPy are two of many open-source packages for scientific computing that use the Python programming language. This website is a portal for all scientific computing with Python, not just NumPy and SciPy. See the index under [Topical Software](#) in the navigation bar.

SciPy is nearing completion of a major overhaul. The new SciPy features high performance for both small and large arrays, and rich syntactic functionality for both. This effort is mostly



**News**

**NumPy 1.0.1 released!!!** (2006-12-02) See the [Download](#) and [Release Notes](#) pages.

**SciPy 0.5.2 released!** (2006-12-07) See the [Download](#) and [Release Notes](#) pages.

**Older news**

**NumPy 1.0 released!!!** (2006-10-25) See the [Download](#) and [Release Notes](#) pages.

**NumPy 1.0rc3 released!** (2006-10-18) See the [Download](#) and [Release Notes](#) pages.

**NumPy 1.0rc2 released!** (2006-10-09) See the [Download](#) and [Release Notes](#) pages.

**SciPy 0.5.1 released!** (2006-09-05) See the [Download](#) and [Release Notes](#) pages.

**NumPy 1.0rc1 released!** (2006-09-20) See the [Download](#) and [Release Notes](#) pages.

**NumPy 1.0b5 released!** (2006-09-04) See the [Download](#) and

# Interactive

```
Terminal — python — 59x19
Oliver:~ chris$ python
ActivePython 2.4.3 Build 11 (ActiveState Software Inc.) bas
ed on
Python 2.4.3 (#1, Apr  3 2006, 18:07:18)
[GCC 3.3 20030304 (Apple Computer, Inc. build 1666)] on dar
win
Type "help", "copyright", "credits" or "license" for more i
nformation.
>>> from numpy import *
>>> x = array([[4,-2],[-1,3]])
>>> linalg.cholesky(x)
array([[ 2.        , -0.        ],
       [-0.5      ,  1.6583124]])
```

# Data access

pressure.csv

	Time, Pressure PSI
1	2001/12/09 00:01:00, 0.143141
2	2001/12/09 00:01:30, 0.143141
3	2001/12/09 00:02:00, 0.143141
4	2001/12/09 00:02:30, 0.143141
5	2001/12/09 00:03:00, 0.143141
6	2001/12/09 00:03:30, 0.143141
7	2001/12/09 00:04:00, 0.143141
8	2001/12/09 00:04:30, 0.143141
9	2001/12/09 00:05:00, 0.143141
10	2001/12/09 00:05:30, 0.143141
11	2001/12/09 00:06:00, 0.143141
12	2001/12/09 00:06:30, 0.143141
13	2001/12/09 00:07:00, 0.143141
14	2001/12/09 00:07:30, 0.143141
15	2001/12/09 00:08:00, 0.175653
16	2001/12/09 00:08:30, 0.143141
17	2001/12/09 00:09:00, 0.143141
18	2001/12/09 00:09:30, 0.175653
19	2001/12/09 00:10:00, 0.143141
20	2001/12/09 00:10:30, 0.143141
21	2001/12/09 00:11:00, 0.143141
22	2001/12/09 00:11:30, 0.175653
23	2001/12/09 00:12:00, 0.143141
24	2001/12/09 00:12:30, 0.175653
25	2001/12/09 00:13:00, 0.175653

Line: 20 Column: 30 Plain Text Tab Size: 3 —

Line: 50 Column: 30 Plain Text Tab Size: 3 —

Terminal — python — 47x15

```
Oliver:~ chris$ python
ActivePython 2.4.3 Build 11 (ActiveState Software Inc.) based on
Python 2.4.3 (#1, Apr 3 2006, 18:07:18)
[GCC 3.3 20030304 (Apple Computer, Inc. build 1
666)] on darwin
Type "help", "copyright", "credits" or "license"
" for more information.

>>> import csv
>>> datafile = open("pressure.csv")
>>> reader = csv.reader(datafile)
>>> for row in reader:
...     print row
```

Terminal — python — 47x15

```
['2002/03/03 13:43:30', '-0.117045']  
['2002/03/03 13:44:00', '-0.117045']  
['2002/03/03 13:44:30', '-0.117045']  
['2002/03/03 13:45:00', '-0.117045']  
['2002/03/03 13:45:30', '-0.117045']  
['2002/03/03 13:46:00', '-0.149579']  
['2002/03/03 13:46:30', '-0.117045']  
['2002/03/03 13:47:00', '-0.117045']  
['2002/03/03 13:47:30', '-0.117045']  
['2002/03/03 13:48:00', '-0.117045']  
['2002/03/03 13:48:30', '-0.149579']  
['2002/03/03 13:49:00', '-0.117045']  
['2002/03/03 13:49:30', '-8.451376E-2']  
['2002/03/03 13:50:00', '']
```

```
>>>
```

```
>>>
```

```
[.5005\03\03 T3:20:00, ...]
```

```
[.5005\03\03 T3:46:30, '-8.4273700E-5, ]
```

```
[.5005\03\03 T3:46:00, '-0.117045, ]
```

re Inc.) based on  
Python 2.4.3 (#1, Apr 3 2006, 18:07:18)  
[GCC 3.3 20030304 (Apple Computer, Inc. build 1  
666)] on darwin  
Type "help", "copyright", "credits" or "license"  
" for more information.  
>>> import MySQLdb  
>>> db = MySQLdb.connect(db="manatee",  
... user="guest",  
... passwd="2phast")  
>>> cur=db.cursor()  
>>> cur.execute("SELECT \* from zones where coun  
ty = 'Volusia'")  
4L  
>>>  
>>>  
>>>  
>>> fλ = „djsuJ0V„()  
>>> cur.execute("SELECT \* from zones where coun

```
Terminal — python — 47x15
... user="guest",
... passwd="2phast")
>>> cur=db.cursor()
>>> cur.execute("SELECT * from zones where coun
ty = 'Volusia'")
4L
>>> cur.fetchall()
((82L, 'Volusia', 'Blue Spring', 1979L, 2L, 197
9L, 11L, 1L, 'seasonal zone', 'NA'), (83L, 'Vol
usia', 'Tomoka River', 1989L, 11L, 1990L, 11L,
1L, 'zone', 'NA'), (84L, 'Volusia', 'St. Johns
River', 1991L, 3L, 1991L, 6L, 1L, 'zone', 'NA')
, (85L, 'Volusia', 'County', 1991L, 7L, 1992L,
1L, 0L, 'zone', '68C-22.012'))
>>>
>>>
JF' 0F', 'zone', '68C-55.015.))
' (82F', 'Volusia', 'County', J66T', L', J66Z'
KIAFL', TATE', BE', Taate', RF', TE', SONG', 'AN')
```

# PyMC

PyMC | Trichech.us

http://trichech.us/?page\_id=3

Ask.com

Atlanta Weather del.icio.us Backpack Flickr Gmail DivShare Trichech.us PyMC + del.icio.us Translate TinyURL

# Trichechus

ecological computing resources

Development Links News Productivity Software

## PyMC

### Markov chain Monte Carlo for Python

Bayesian estimation, particularly using Markov chain Monte Carlo (MCMC), is an increasingly relevant approach to statistical estimation. However, few statistical software packages implement MCMC samplers, and they are non-trivial to code by hand. PyMC is a python module that implements the Metropolis-Hastings algorithm as a python class, and is extremely flexible and applicable to a large suite of problems. PyMC includes methods for summarizing output, plotting, goodness-of-fit and convergence diagnostics.

About

Downloads

PyMC

KIVA

What's del.icio.us

» www.saysosoft.com

Annotation is an easy to use application that allows you to code events in terms of their type, onset, and duration

» Where to Publish

A summary of target journals for ecological publications.

» uDIG

Generality  
Simplicity

- Data access and manipulation
- Parameter and node initialization
- Specification of conditional posterior

# Adaptive Random Walk Metropolis-Hastings

$$\theta^{(t+1)} \quad = \quad \theta^{(t)} + \epsilon_t$$

$$\epsilon_t \quad \sim \quad f(\phi)$$

# Symmetric

$$q(\theta' | \theta^{(t)}) = q(\theta^{(t)} | \theta')$$

$$a(\theta', \theta) = \frac{\pi(\theta')}{\pi(\theta)}$$

$$\theta^{(t+1)} \quad = \quad \theta^{(t)} + \epsilon_t$$

$$\epsilon_t \quad \sim \quad f(\phi)$$

# Likelihood Methods

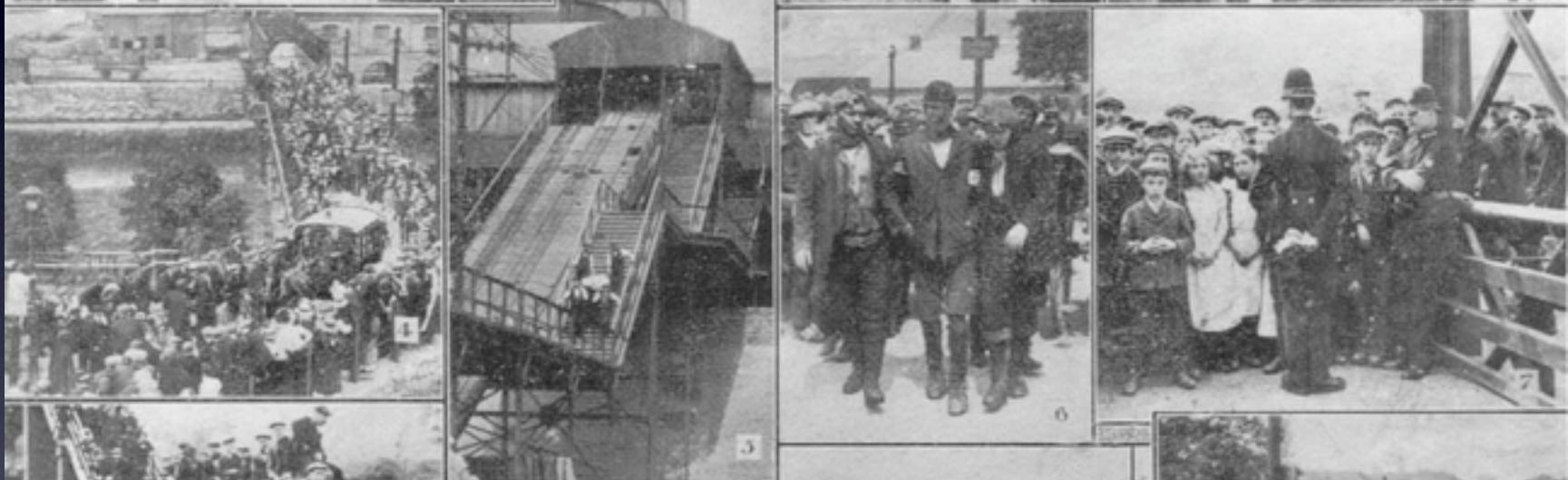
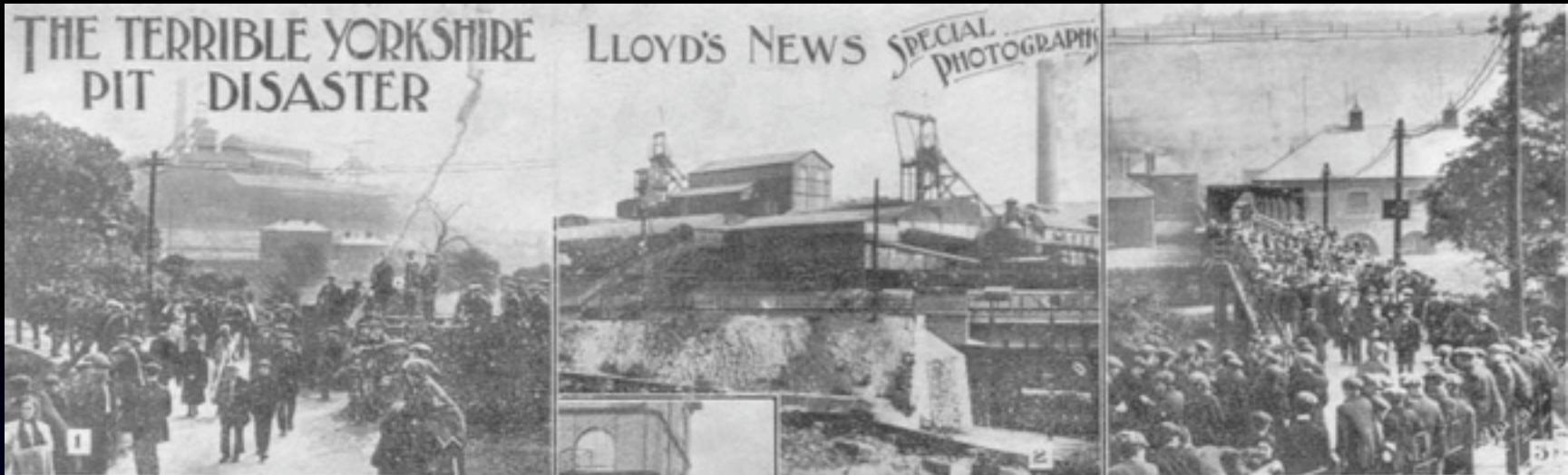
bernoulli\_like  
beta\_like  
binomial\_like  
categorical\_like  
cauchy\_like  
chi2\_like  
dirichlet\_like  
gamma\_like  
geometric\_like  
half\_normal\_like  
hypergeometric\_like  
inverse\_gamma\_like

lognormal\_like  
multinomial\_like  
multivariate  
hypergeometric\_like  
multivariate\_normal\_like  
negative\_binomial\_like  
normal\_like  
poisson\_like  
uniform\_like  
uniform\_mixture\_like  
weibull\_like  
wishart\_like

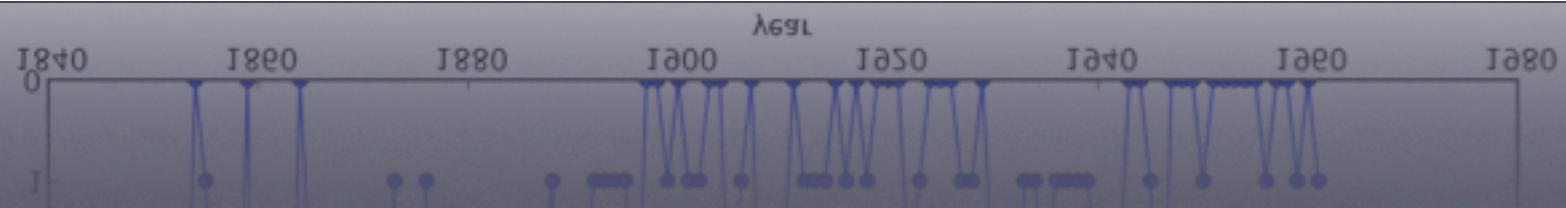
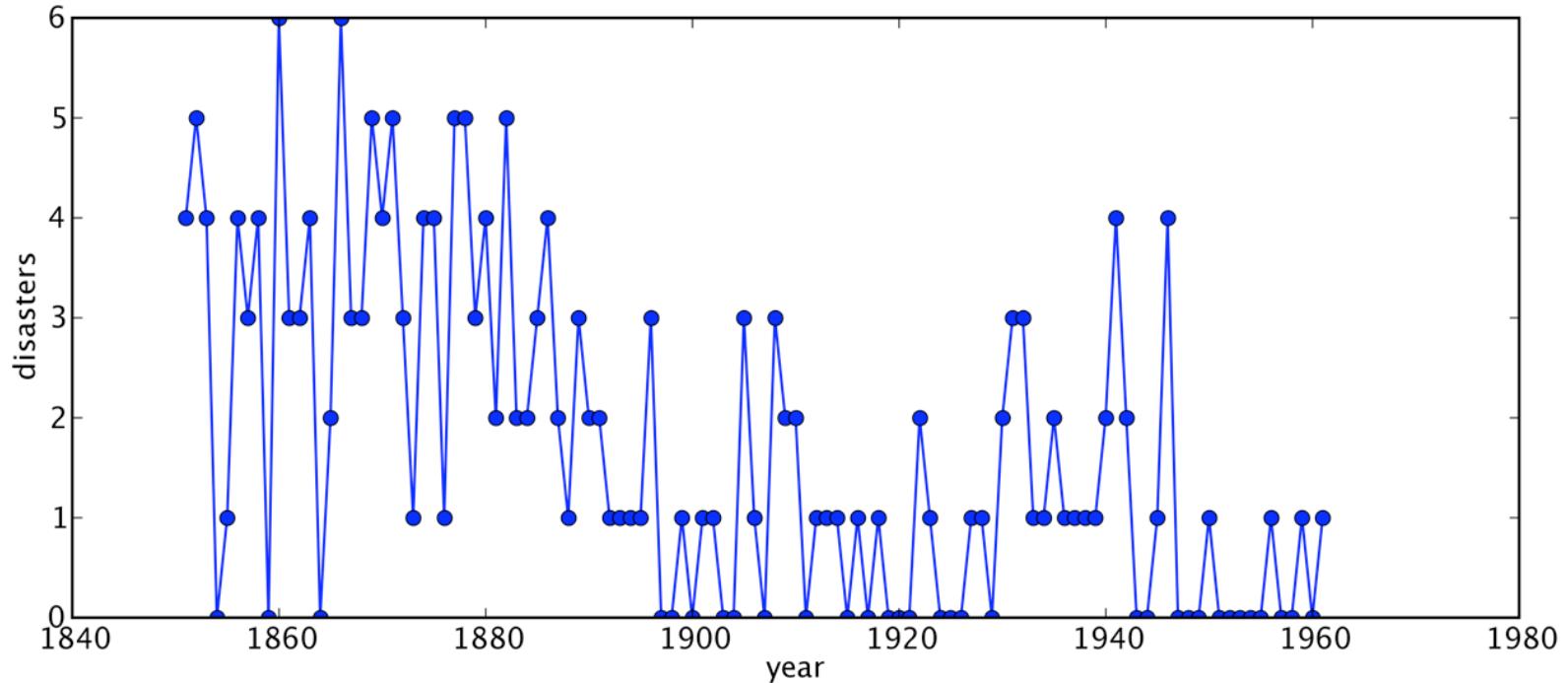
# Prior methods

# THE TERRIBLE YORKSHIRE PIT DISASTER

LLOYD'S NEWS *SPECIAL  
PHOTOGRAPHS*



# Coal Mining Disasters



$$\begin{aligned} X_{t \leq k} &\sim \text{Poisson}(\lambda_1) \\ X_{t > k} &\sim \text{Poisson}(\lambda_2) \\ k &\sim \text{Uniform}(1851, 1962) \end{aligned}$$

```
1 class DisasterSampler(MetropolisHastings):
2
3     def __init__(self):
4         """Class initialization"""
5
6         MetropolisHastings.__init__(self)
7
8         # Sample changepoint data (Coal mining disasters per year)
9         self.data = (4, 5, 4, 0, 1, 4, 3, 4, 0, 6, 3, 3, 4, 0, 2, 6,
10            3, 3, 5, 4, 5, 3, 1, 4, 4, 1, 5, 5, 3, 4, 2, 5,
11            2, 2, 3, 4, 2, 1, 3, 2, 2, 1, 1, 1, 1, 3, 0, 0,
12            1, 0, 1, 1, 0, 0, 3, 1, 0, 3, 2, 2, 0, 1, 1, 1,
13            0, 1, 0, 1, 0, 0, 0, 2, 1, 0, 0, 0, 1, 1, 0, 2,
14            3, 3, 1, 1, 2, 1, 1, 1, 2, 4, 2, 0, 0, 1, 4,
15            0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1)
16
17         # Switchpoint is specified as a parameter to be estimated
18         self.parameter(name='k', init_val=50, discrete=True)
19
20         # Rate parameters of poisson distributions
21         self.parameter(name='theta', init_val=array([1.0, 1.0]))
22
```

Line: 30 Column: 31 Python

Soft Tabs:

4

calculate\_likelihood(self)

Line: 30 Column: 31 Python

Soft Tabs:

4

calculate\_likelihood(self)

MCMC.py

```
10
11     # Change point
12     self.parameter(name='k', init_val=50, discrete=True)
13
14     # Rate parameters of poisson distributions
15     self.parameter(name='theta', init_val=array([1.0, 1.0]))
16
17 def model(self):
18     """Specification of joint posterior"""
19
20     # Constrain k with prior
21     self.uniform_prior(self.k, 1, len(self.data)-2)
22
23     # Joint likelihood of parameters based on 2 assumed
24     # Poisson densities
25     self.poisson_like(self.data[:self.k], self.theta[0],
26     name='early_mean')
27
28     self.poisson_like(self.data[self.k:], self.theta[1],
29     name='late_mean')
```

Line: 21 Column: 56 Python

Line: 51 Column: 26 Python

detection.py

```
152 def run(iterations=10000, burn=5000, thin=1):
153     # Run MCMC sampler
154
155     # Instantiate sampler
156     sampler = DisasterSampler()
157
158     # Sample
159     sampler.sample(iterations=iterations,
160     .           burn=burn, thin=thin)
161
162     # Run convergence diagnostics
163     sampler.convergence(burn=burn, thin=thin)
164
165     # Plot ACF
166     sampler.autocorrelation(burn=burn, thin=thin)
167
168     # Goodness-of-fit
169     sampler.goodness(iterations/10, burn=burn,
170     thin=thin)
```

Line: 156 Column: 23 Python Soft Tabs: 4 run(iterations, burn, thin)  
Line: 156 Column: 53 Python Soft Tabs: 4 run(iterations, burn, thin)

Terminal — python — 58x18

Iteration 1500 at 19.3542010784

Tuning theta

current value: [ 3.31502424 1.01090971]  
current proposal hyperparameter: 0.3486784401  
acceptance rate: 0.27  
adaptive scaling factor: 0.3486784401  
new proposal hyperparameter: 0.3486784401

Tuning k

current value: 37  
current proposal hyperparameter: 4.715895382  
acceptance rate: 0.38  
adaptive scaling factor: 4.715895382  
new proposal hyperparameter: 4.715895382

Node deviance current value: 347.111143224



Node deviance current value: 347.111143224

Terminal — python — 58x18

Tuning k

current value: 40  
current proposal hyperparameter: 4.715895382  
acceptance rate: 0.46  
adaptive scaling factor: 4.715895382  
new proposal hyperparameter: 4.715895382

Node deviance current value: 348.765061729

\*\*\* Finished tuning proposals \*\*\*

Iteration 1900 at 24.9736809731

Iteration 2000 at 26.2398450375

Iteration 2100 at 27.4188439846

Iteration 2200 at 28.6151781082

Iteration 2500 at 28.6151781082

## Terminal — bash — 58x18

Node: theta

95% HPD interval = [[ 2.60180233 3.71626677]  
[ 0.72512715 1.19220731]]

mc error = [ 0.00579903 0.00243867]

mean = [ 3.11459681 0.94737281]

n = 2500

quantiles =

2.5: [ 2.58258839 0.72033762]

25: [ 2.90800165 0.86124055]

50: [ 3.09468546 0.9436626 ]

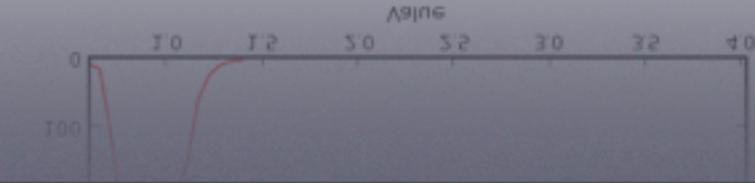
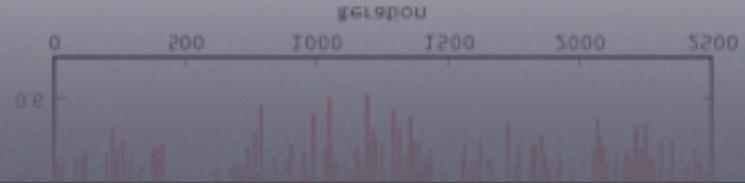
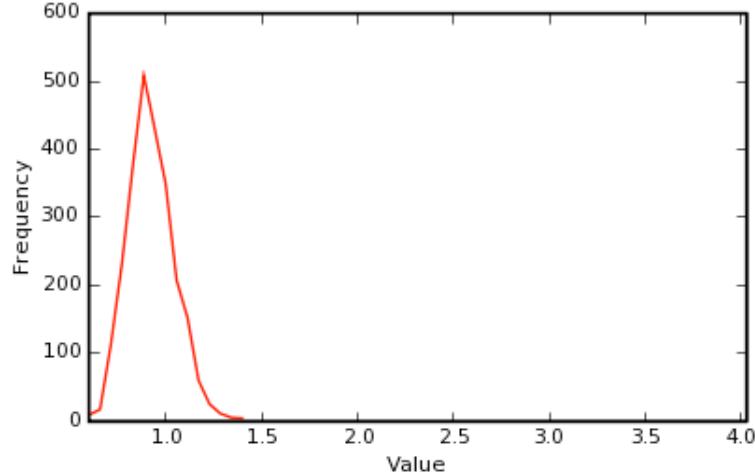
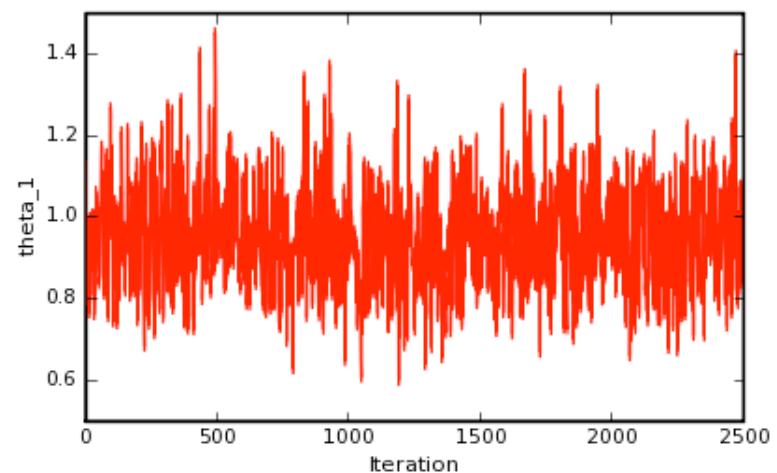
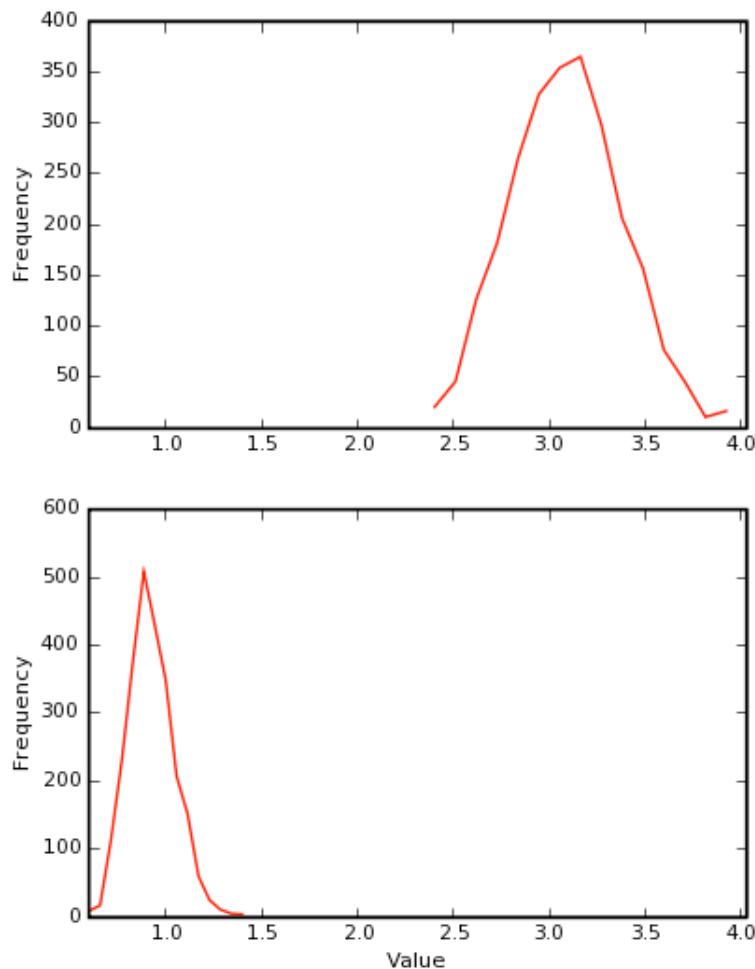
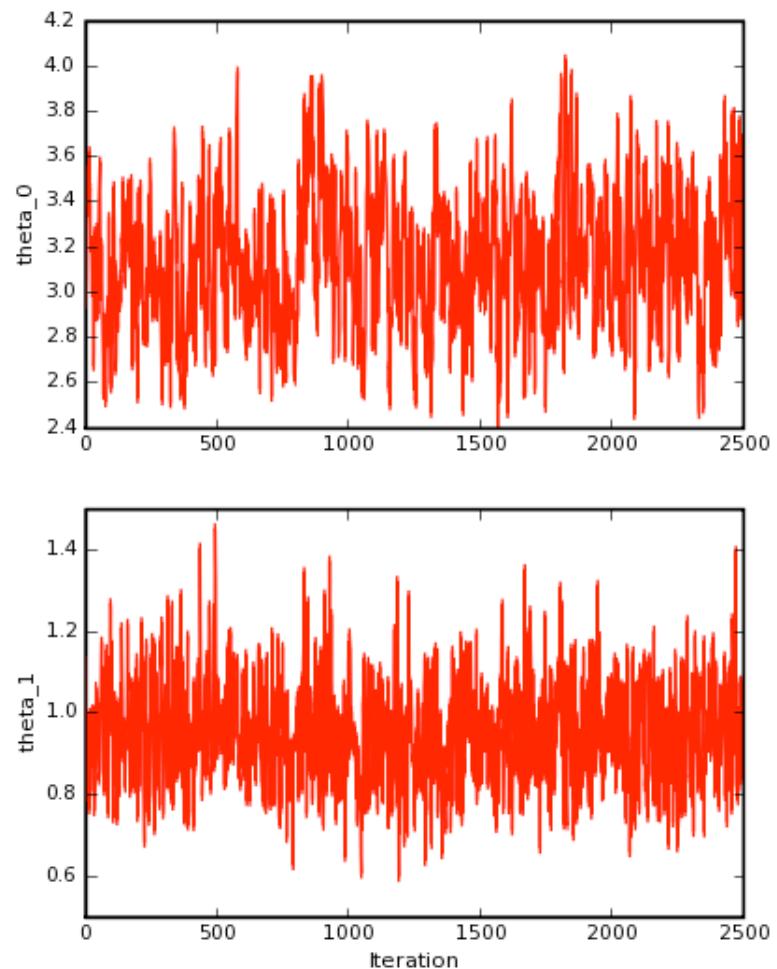
75: [ 3.31816395 1.03255296]

97.5: [ 3.71415256 1.18992997]

standard deviation = [ 0.2899514 0.1219333]

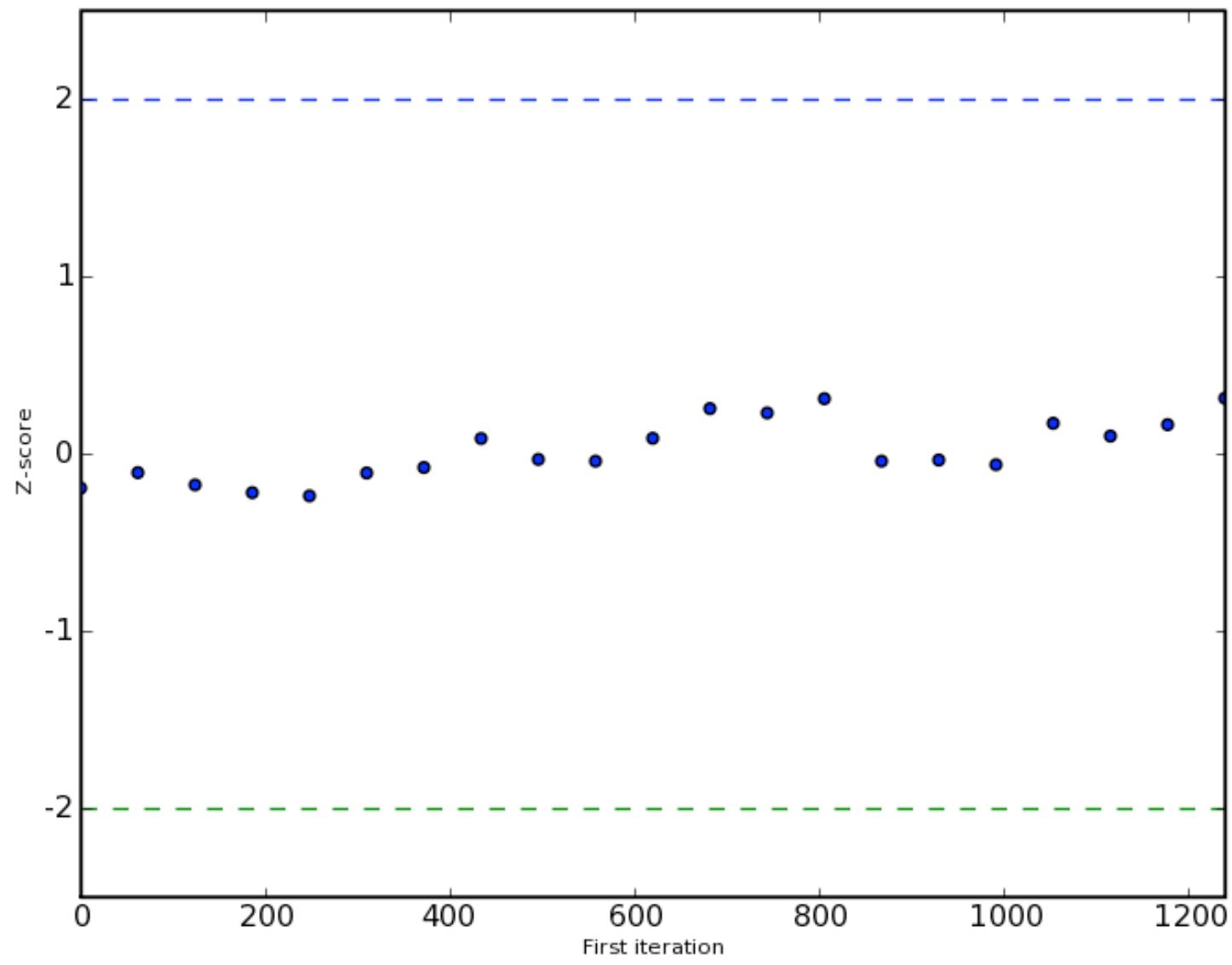
---

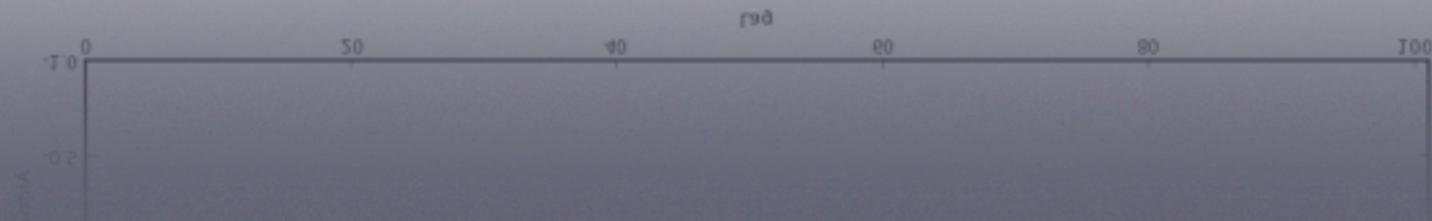
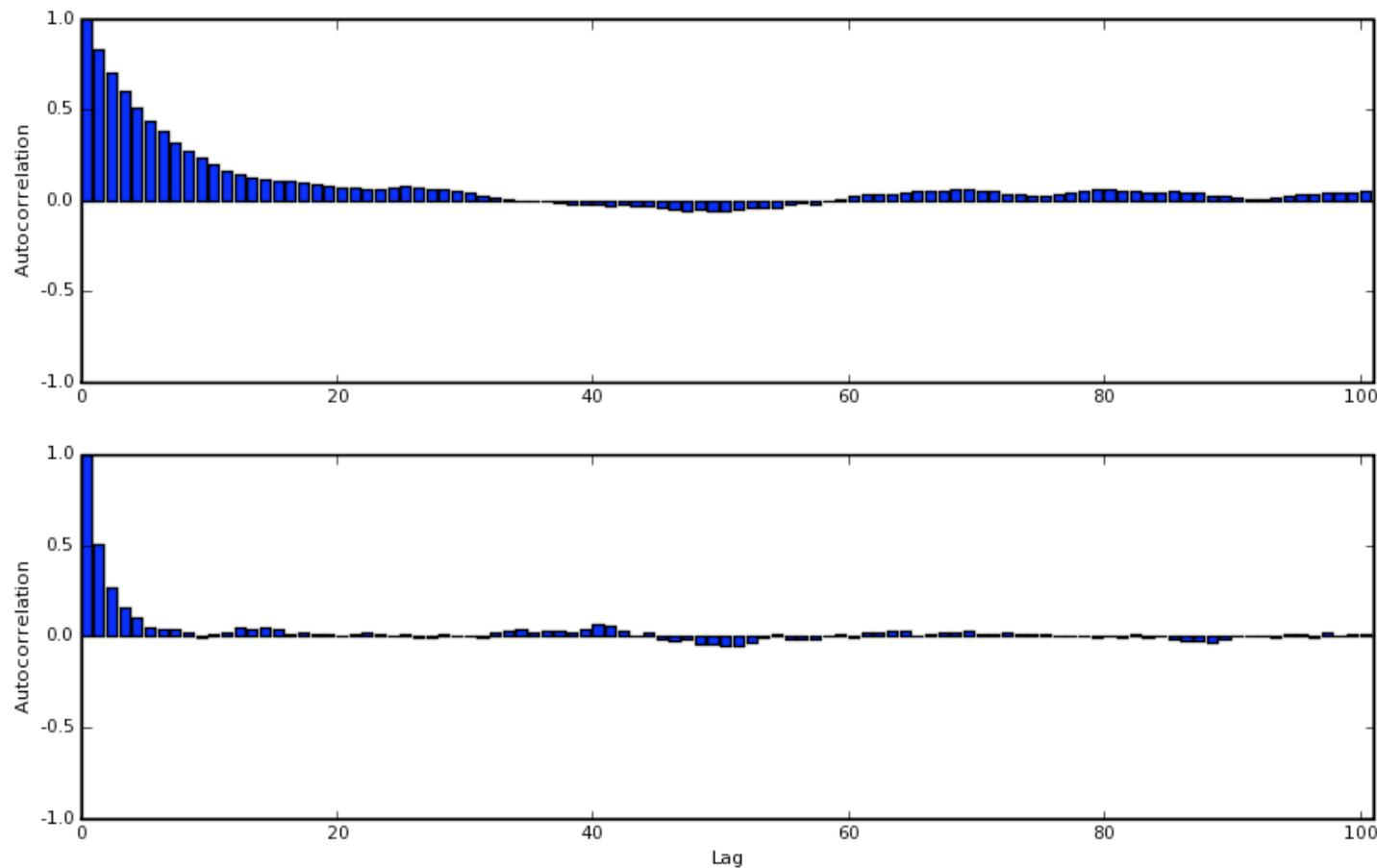
DIC = 351.640220678

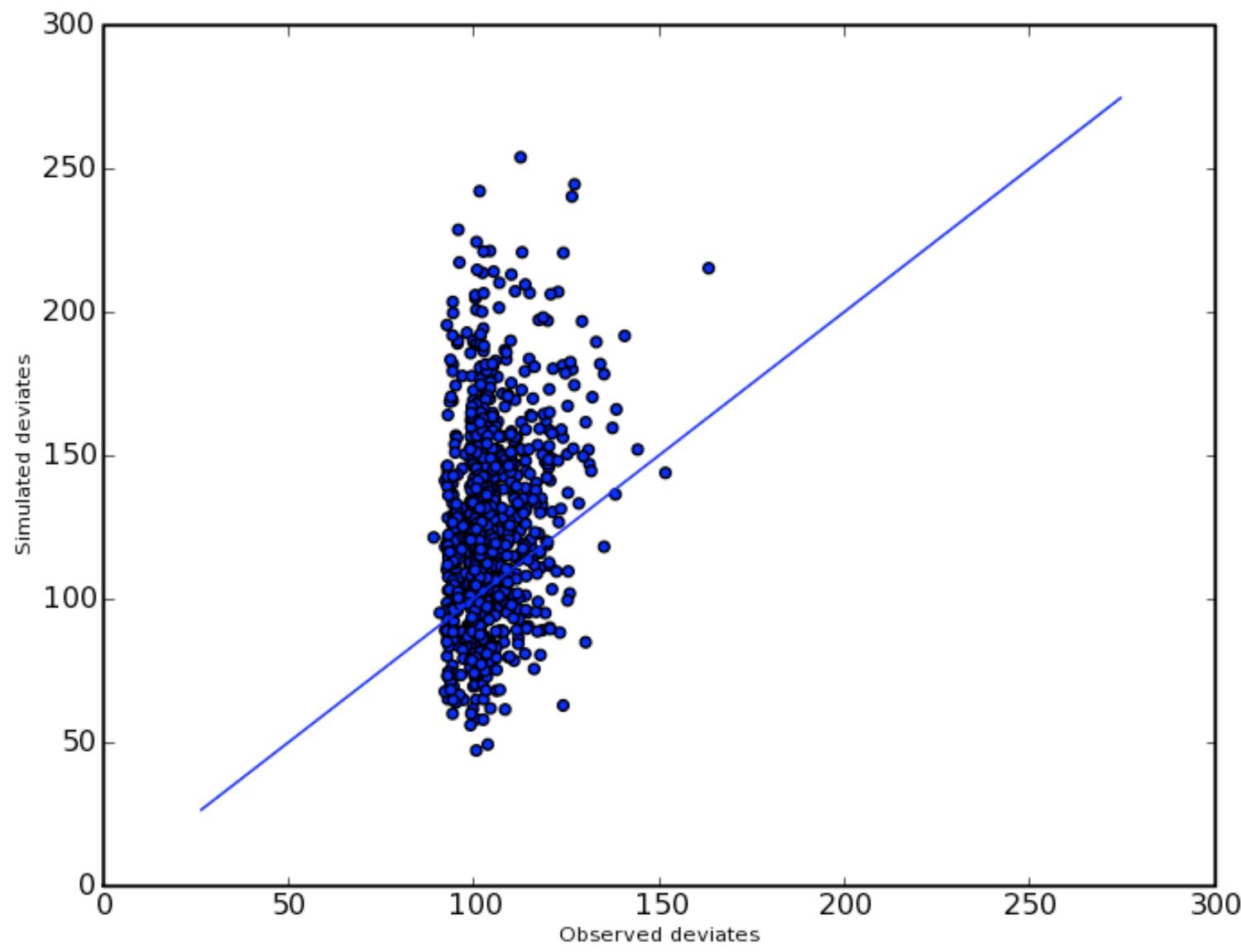


# Diagnostics

(or, how to know when you're done)







Scatterplot of simulated vs observed

0 20 100 200 500 520 300

# User's Guide

## Chapter 2

## Installation

PyMC is known to run on Mac OS X, Linux and Windows, but in theory should be able to work on just about any platform for which Python is available (and there are many). Rather than a standalone application, PyMC is simply a *module* for the Python programming language. That is, a set of programming classes that are imported into the Python programming environment for subsequent use. The classes are very general, and can therefore be implemented in a variety of Bayesian analytic tasks. I will describe how to install PyMC using both binary installers (for Mac and Windows) and builds from source code.

Pre-built binary and source distributions of PyMC are available from [trichech.us](#). There are three PyMC distribution formats provided, each containing the same PyMC module, but targeted for different computing platforms. For Macintosh (OS X) users there is an installer package (.mpkg), for Linux users a compressed tar (.tar.gz) source archive, and for Microsoft Windows a binary executable installer (note that this may not always be the latest version, as PyMC is not specifically developed or tested for Windows).

PyMC requires some prerequisite packages to be present on the system before the PyMC package itself is installed. Fortunately, there are currently only a few dependencies, and all are freely available online.

- Python version 2.4 or later. I recommend using the [ActiveState distributions](#) for Mac OS X and Linux. Windows users should download and install [Enthought Python](#), which contains all of the required packages below. Additionally, the Mac OS X binary distribution of PyMC comes bundled with these prerequisites.
- FORTRAN and C compilers, preferably G77 and GCC (*i.e.* the GNU compilers), respectively. You may use other compilers, but most have not been tested with PyMC.
- [Numpy](#), latest version. This is a fundamental scientific programming package for python, which replaces a slew of stand-alone packages (including Numeric) that needed to be installed in the past.
- [Matplotlib](#) version 0.96 or later. This package is used for plotting of MCMC output.

# CMR Models in PyMC

# CLOSED CR MODELING

- Based on Huggins conditional likelihood
- Heterogeneity modeled via logit-normal distribution of individual random effects
- N is a derived parameter
- Half-normal prior (precision = 1/10<sup>6</sup>) on SD of random effects
- Tested with simulated data:  
 $N=100, p=c=0.3, k=5, CV_h=0.10$

# PYMC code

## *Priors and random effects*

PRIOR on precision

```
self.half_normal_prior(sd, 1./b**2)
```

RANDOM EFFECT DISTRIBUTION

```
self.normal_like(self.indiv, 0., self.tau)
```

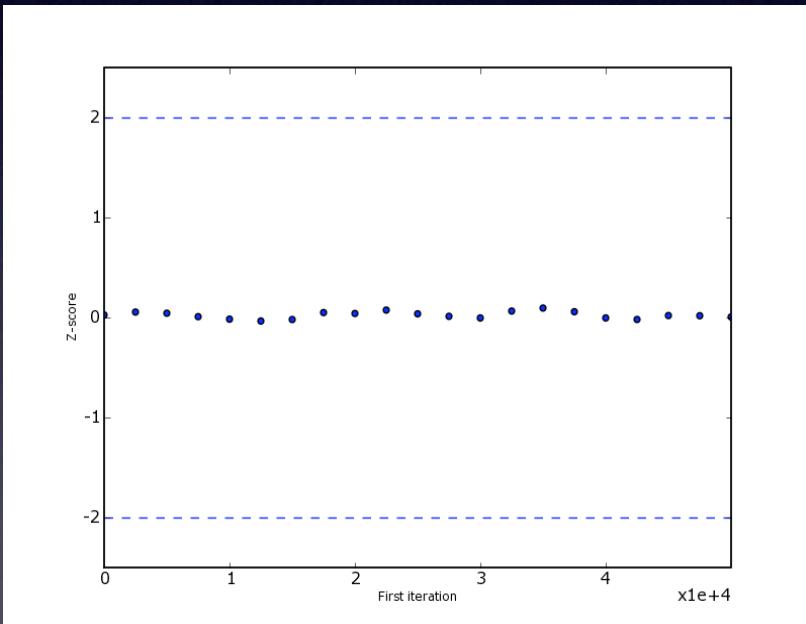
closed.py

```
75      # Loop over capture histories
76      for animal, hist in enumerate(data):
77
78          # capture frequencies of ith history
79          z = data[hist]
80
81          # Parse capture histories into tuple
82          catch = tuplestr(hist)
83
84          # From Huggins, 1991, p.726
85          pstar = invlogit(self.beta_0 +
86                           animal_effect[animal])
87          pb = [(1. - pstar)**(self.k - j) for j in
88                           range(self.k)]
89
90          gamma = [pstar / (1. - (1. - (sum(catch[:j]) > 0))
91                           * x) for j, x in enumerate(pb)]
92          self.bernoulli_like(catch, gamma, factor=z)
93
94          # add estimated abundance for hist i; observed
95          # freq/(P not observing i)
96          self.Nhat += z/(1. - pb[0])
```

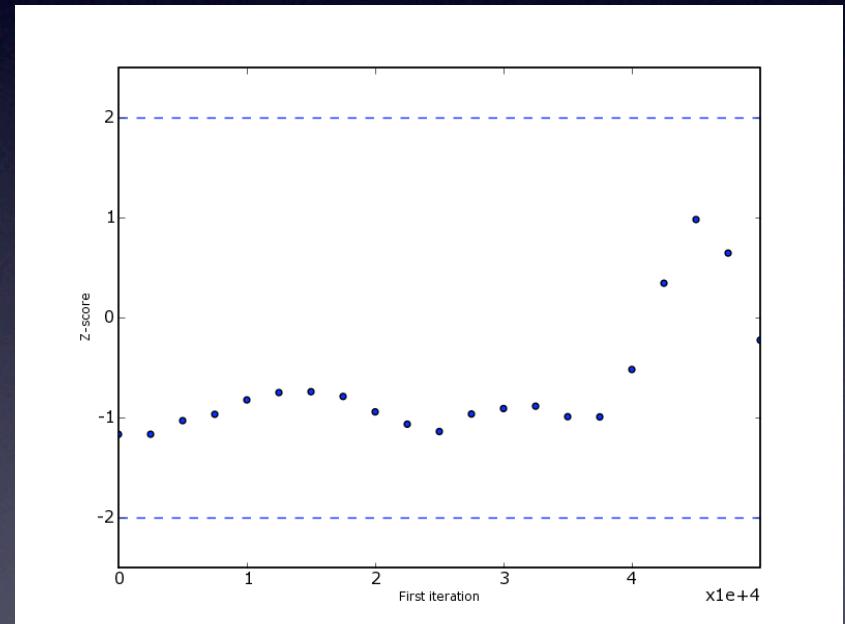
Line: 10 Column: 18 Python Soft Tabs: 4 —  
Line: 10 Column: 18 Ќ Python Soft Tabs: 4 —

# Convergence diagnostics

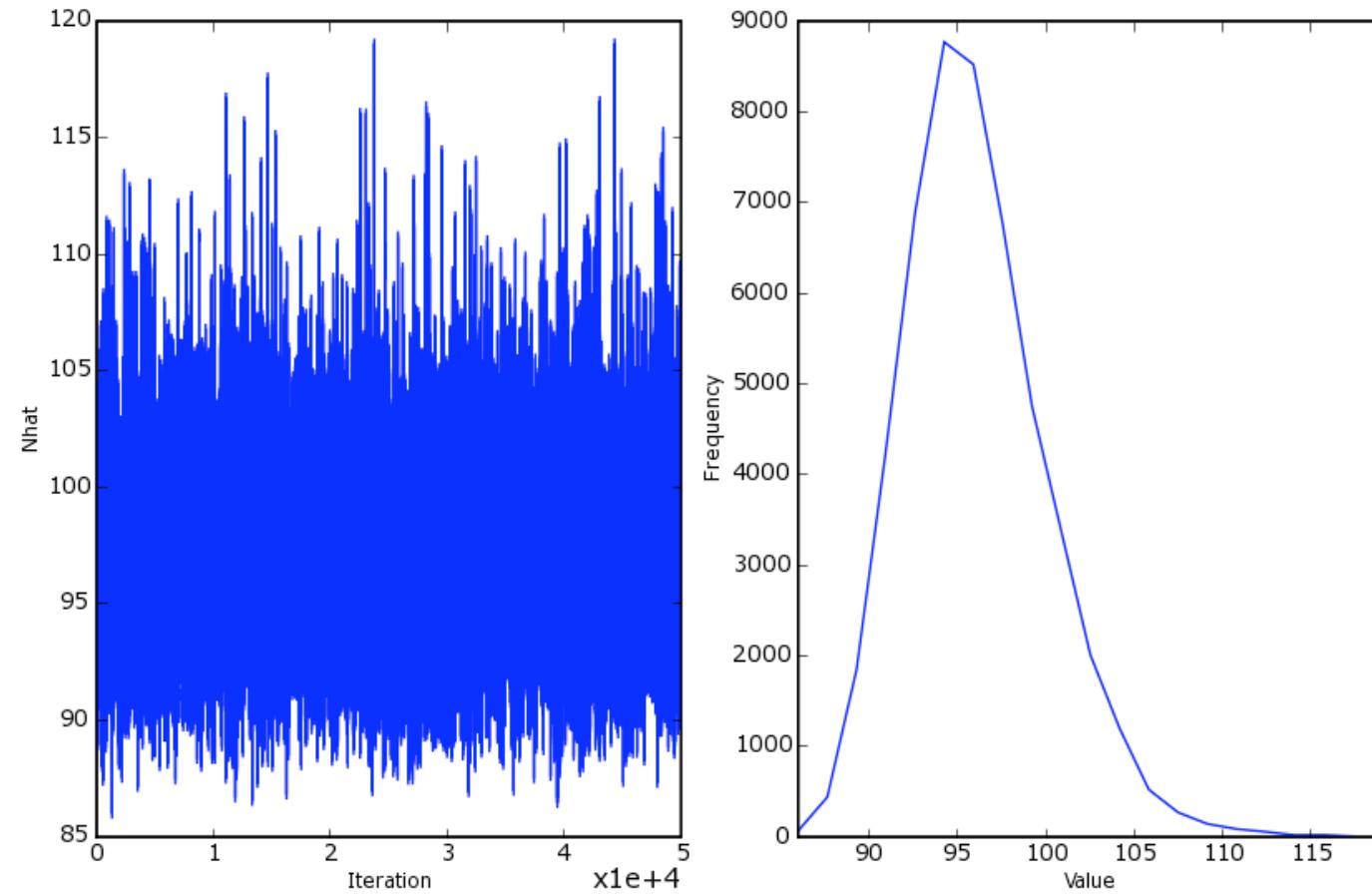
$\beta_0$



$\tau$

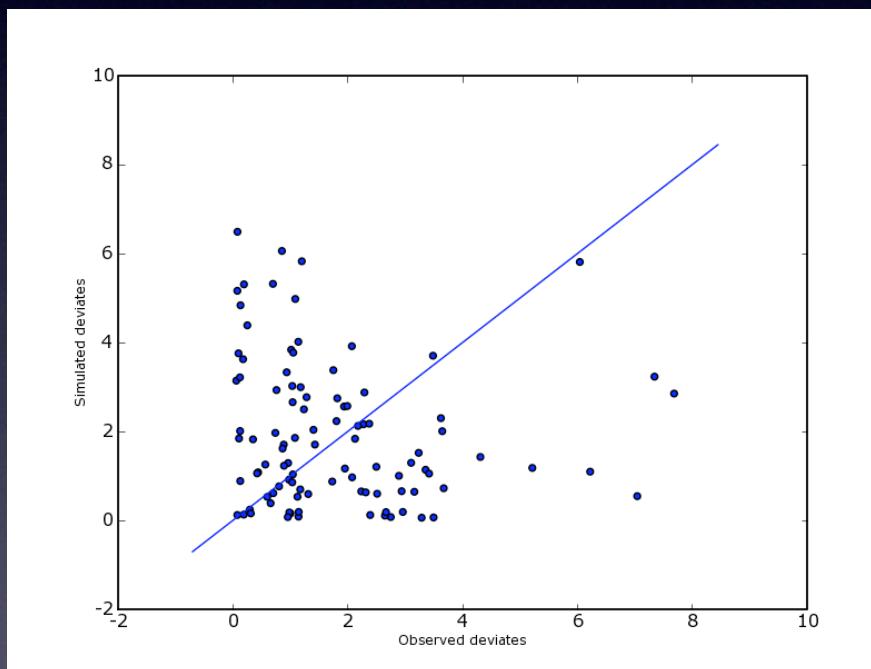


# Trace and posterior of $\hat{N}$

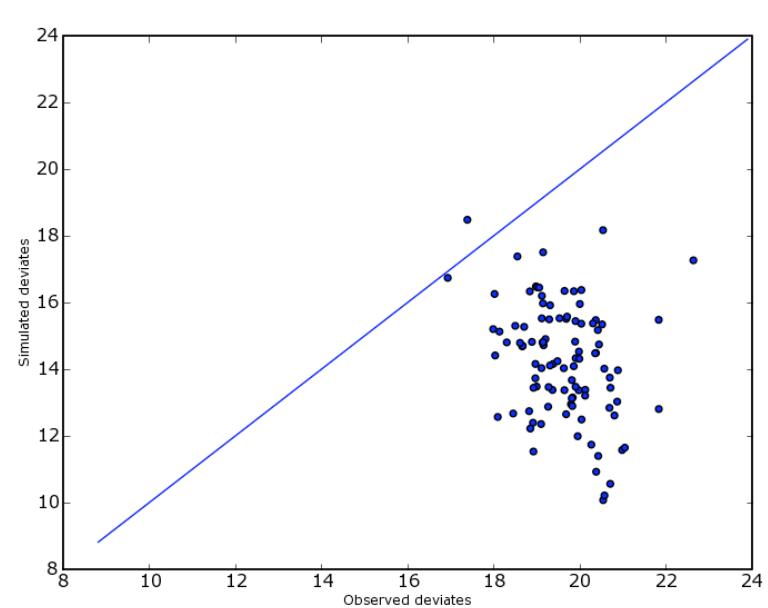


# Goodness of Fit

Normal likelihood  
(random effect)



Bernoulli likelihood



# OPEN ESTIMATION

- Ring-recovery example for American black ducks
- Age-, sex-, and time- specific Brownie parameterization
- Incorporation of additive vs. compensatory mortality via an ‘ultrastructural’ approach

```
153 def model(self):
154
155     # Half-normal likelihood for tau
156     self.half_normal_prior(1/sqrt(self.tau_S), 1000**-2)
157
158     # Constraint b1 to be negative
159     self.uniform_prior(self.b1,-1000.,-0.00001)
160
161     # Random effect
162     self.normal_like(self.re_s,0.,self.tau_S)
163
164 S=resize([0.0]*sexes*ages*years,[sexes,ages,years])
165
166 for s in range(sexes):
167     for a in range(ages):
168         for y in range(years):
169             try:
170                 S[s,a,y] =
171                 invlogit(logit(self.ss[s,a])+self.b1*self.f[s,a,y]+self.re_s[y])
172             except ValueError:
173                 return -inf
```

survival.py

```
68
69      # Loop over sexes, ages and years
70      for s in range(sexes):
71          for a in range(ages):
72              for y in range(years):
73
74                  # Build recovery probabilities
75                  pi = [self.f[s, a, y]]
76
77                  pi += [(S[s, young, y]*a or S[s, adult,
78 .           y])*product(S[s, adult, y+1:ry]) * self.f[s, adult, ry] for ry
79 .           in range(y+1, years+1)]
80
81                  obs_X = X[s, a, y, y:]
82
83                  # Multinomial likelihood of recoveries
84                  self.multinomial_like(obs_X, RR[s, a, y],
85 .                   pi, name = 'surv')
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
```

Line: 143 Column: 1 Python

Line: 143 Column: 1 Python

Soft Tabs: 4 model(self)

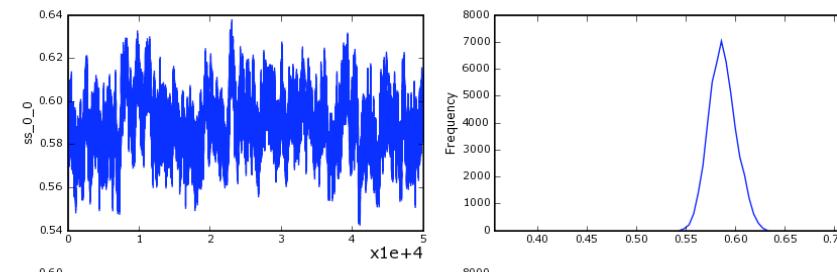
Soft Tabs: 4 model(self)

83

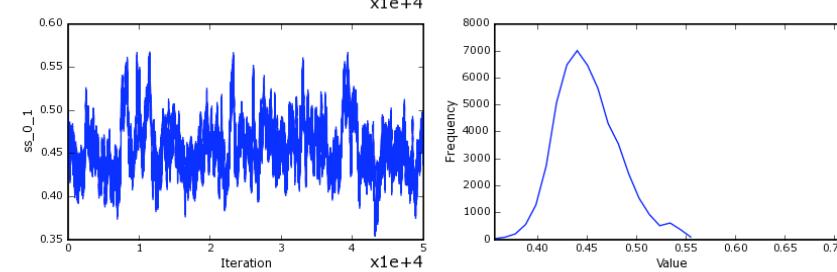
br, name = 'surv')

# Additive model

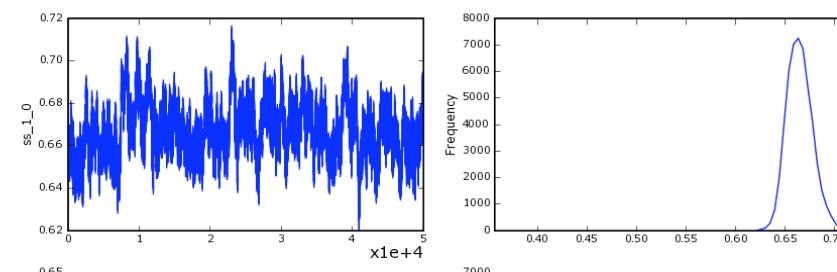
$S_{af}$



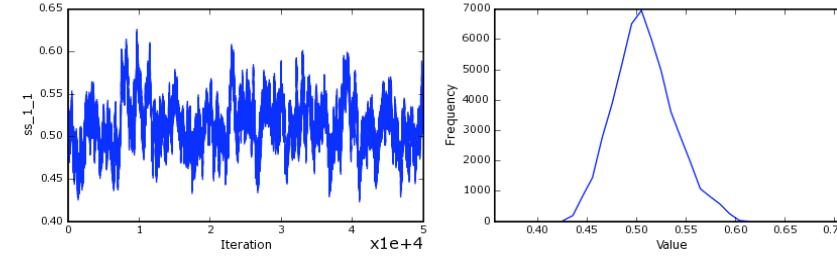
$S_{jf}$



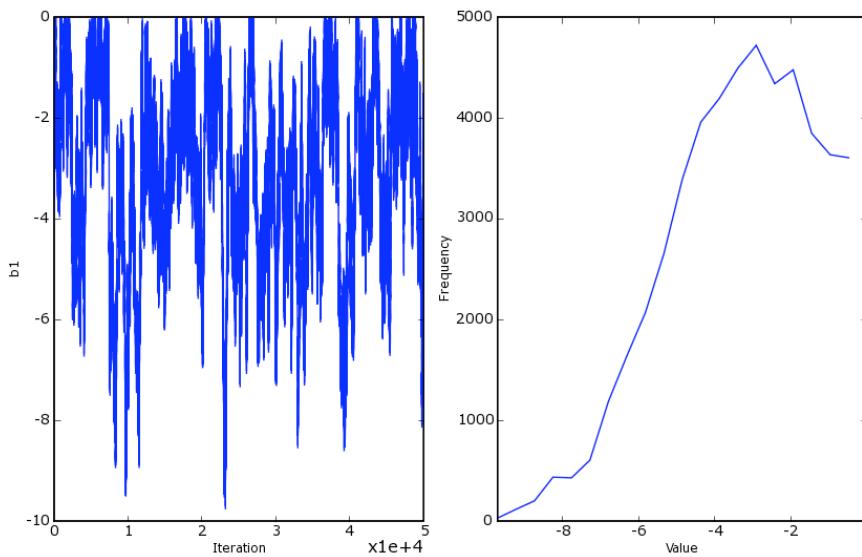
$S_{am}$



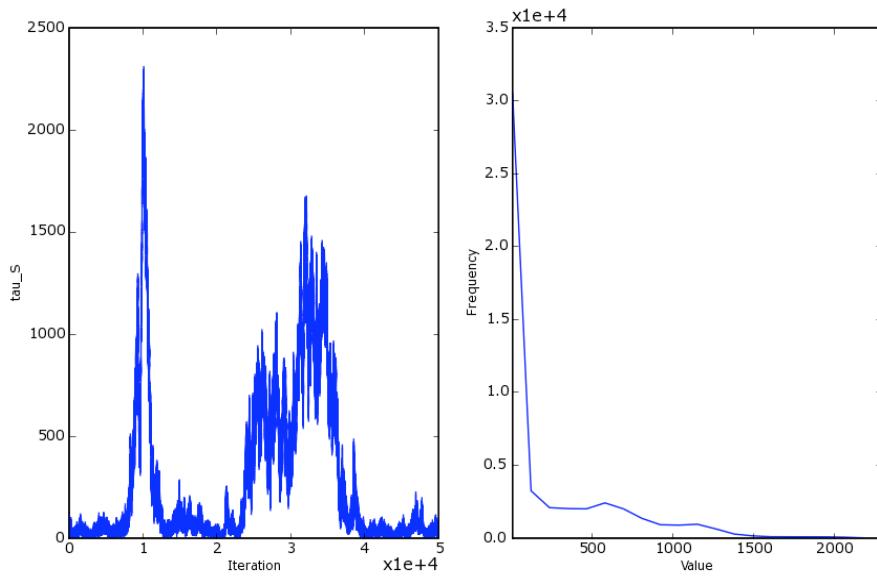
$S_{jm}$



$b_I$



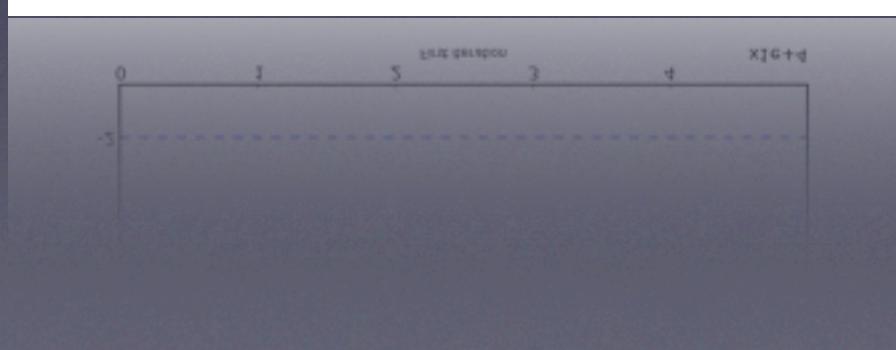
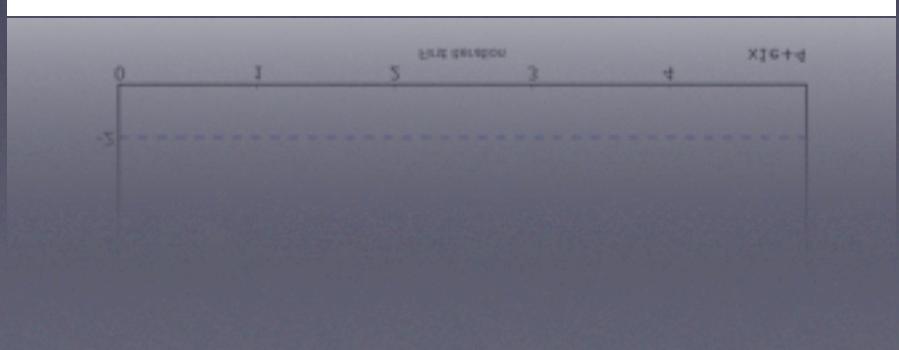
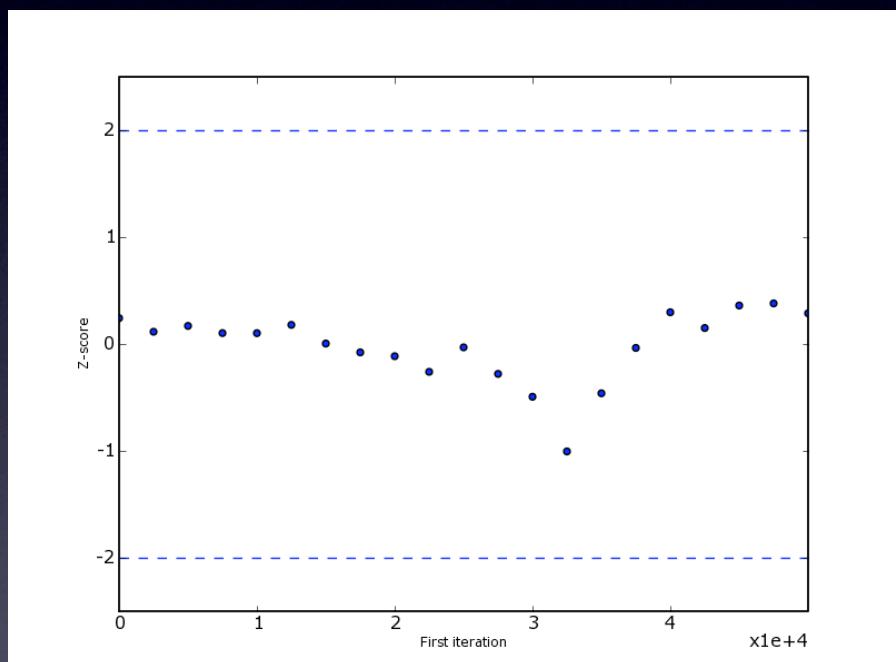
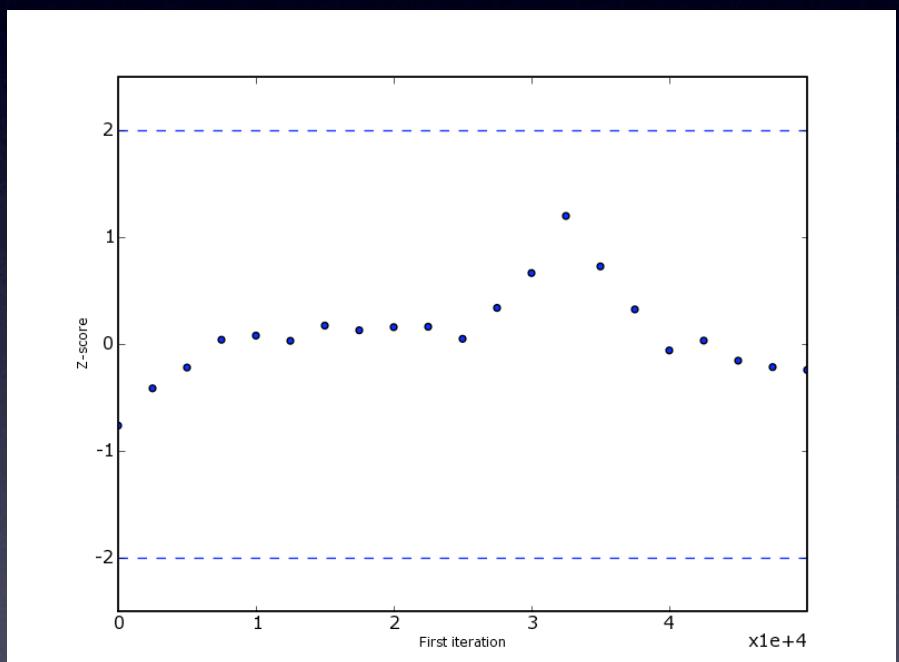
$\tau_s$



# Convergence Diagnostics (100/50K)

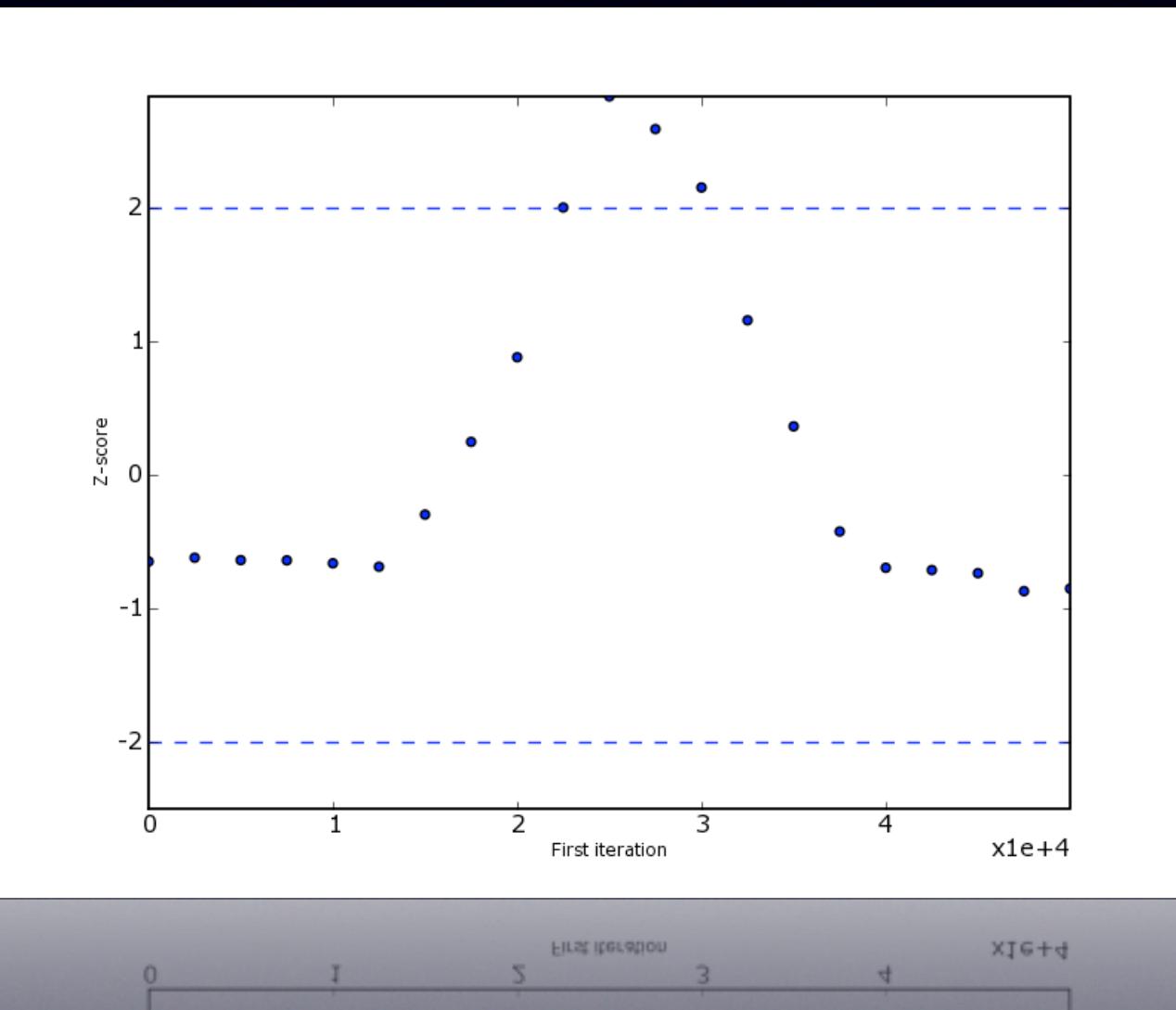
$S_{JF}$

$b_I$



# Convergence Diagnostics

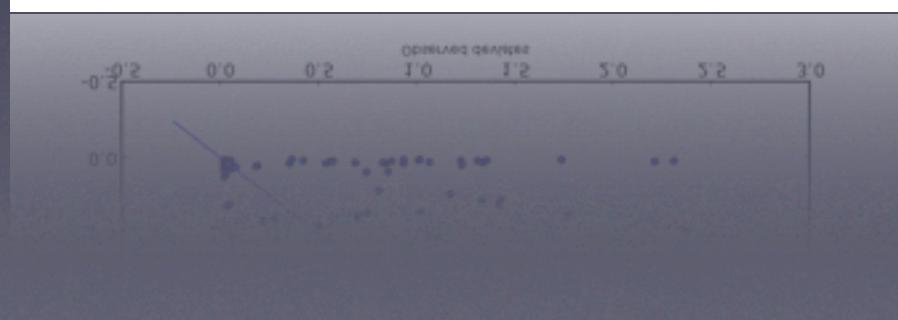
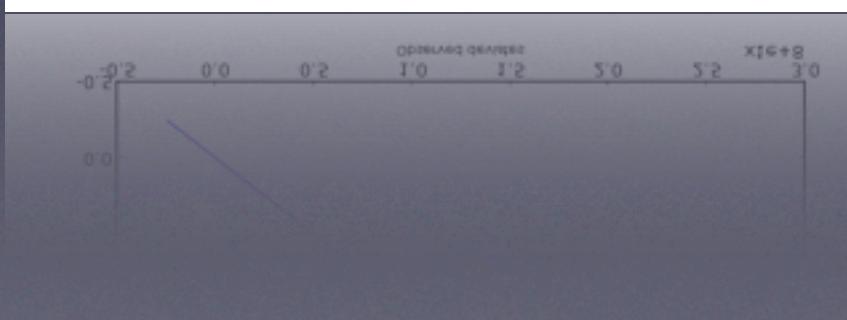
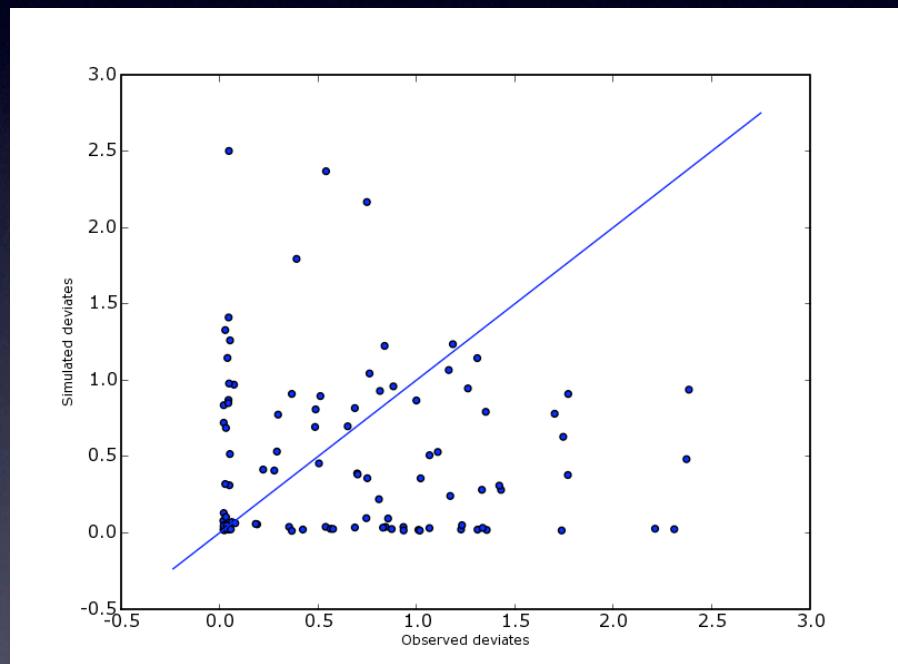
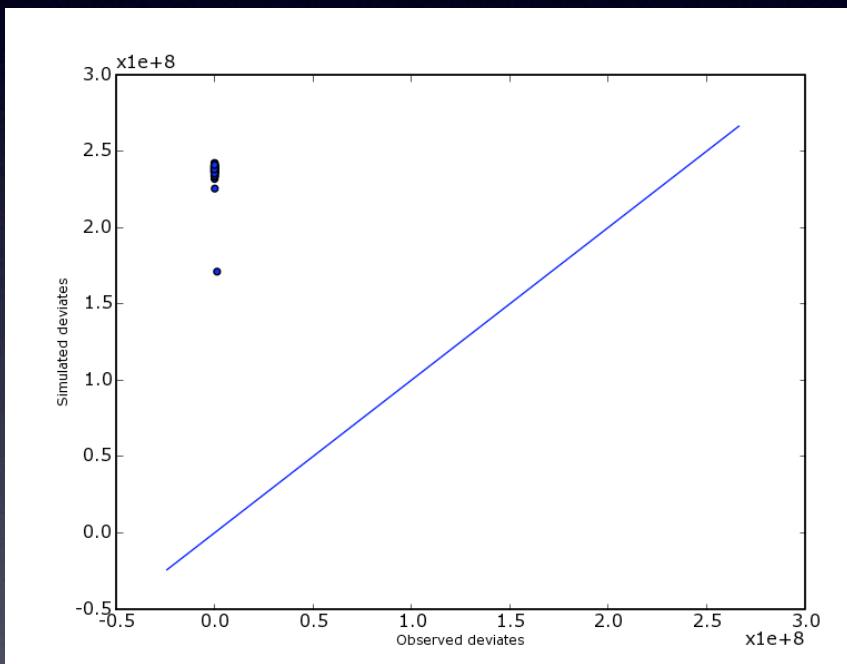
$\tau_s$



# GOF

Multinomial likelihood

Normal likelihood  
(random effect)



# PyMC

- Free and open
- Easy
- Modular
- Extensible
- Reasonably fast

# Future development

David Huard  
Anand Patil

Parallel chains  
RJMCMC  
SQL backend  
Specialized modules

<http://trichech.us>

[http://code.google.com/  
p/pymc](http://code.google.com/p/pymc)