Approximation Methods

SciPy 2011

Maximum *a posteriori* (MAP)

$$\hat{\theta}_{MAP} = \arg\max P(\theta|y)$$

$$P(\theta|y) = \frac{P(y|\theta)P(\theta)}{P(y)}$$

Bayes Formula

$$P(\theta|y) = \frac{P(y|\theta)P(\theta)}{P(y)}$$

Bayes Formula

$$\Rightarrow \log P(\theta|y) = \log P(y|\theta) + \log P(\theta) + C$$

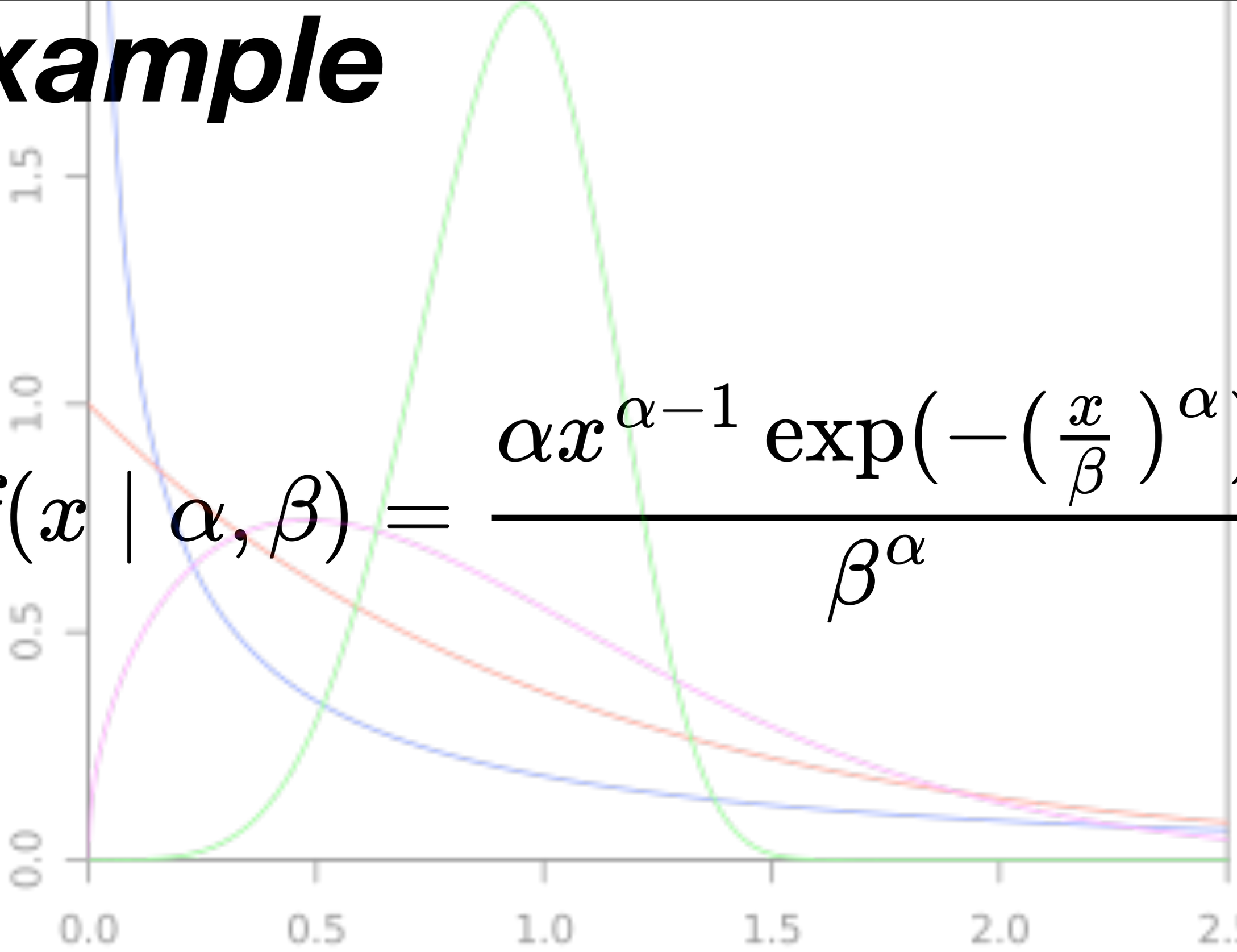
$$P(\theta|y) = \frac{P(y|\theta)P(\theta)}{P(y)}$$

Bayes Formula

$$\Rightarrow \log P(\theta|y) = \log P(y|\theta) + \log P(\theta) + C$$

$$\Rightarrow \hat{\theta}_{MAP} = \arg\max[\log P(y|\theta) + \log P(\theta)]$$

example



$$f(x \mid \alpha, \beta) = \frac{\alpha x^{\alpha-1} \exp\left(-\left(\frac{x}{\beta}\right)^{\alpha}\right)}{\beta^{\alpha}}$$

Weibull distribution

weibull.py

```
1  import pymc
2
3  # Some fake data
4  alpha = 3
5  beta = 5
6  N = 100
7  dataset = pymc.rweibull(alpha,beta, N)
8
9  # Model
10 a = pymc.Uniform('a', lower=0, upper=10, value=5, doc='Weibull alpha
   parameter')
11 b = pymc.Uniform('b', lower=0, upper=10, value=5, doc='Weibull beta
   parameter')
12 like = pymc.Weibull('like', alpha=a, beta=b, value=dataset, observed=True)
```

Line: 12 Column: 42 Python Soft Tabs: 4

```
>>> M = pymc.MAP(weibull)
>>> M.fit()
>>> M.a.value
array(3.2281850543716857)
>>> M.b.value
array(4.7273717106950723)
```

```
N.fit(method='fmin', iterlim=1000, tol=.
001):

Causes the normal approximation object to
fit itself.

method: May be one of the following, from
the scipy.optimize package:
     -fmin_l_bfgs_b
     -fmin_ncg
     -fmin_cg
     -fmin_powell
     -fmin
```

weibull.py

```
1  import pymc
2
3  # Some fake data
4  alpha = 3
5  beta = 5
6  N = 100
7  dataset = pymc.rweibull(alpha,beta, N)
8
9  # Model
10 a = pymc.Uniform('a', lower=0, upper=10, value=5, doc='Weibull alpha
   parameter')
11 b = pymc.Uniform('b', lower=0, upper=10, value=5, doc='Weibull beta
   parameter')
12 like = pymc.Weibull('like', alpha=a, beta=b, value=dataset, observed=True)
```

Line: 12 Column: 42 Python Soft Tabs: 4

```
>>> M.AIC
369.57863603595712
>>> M.BIC
374.78897640793332
```

Normal approximation

$$P(\theta|y) \approx N(E(\theta|y), C(\theta|y))$$

Taylor expansion

$$\log f(\theta|y) \approx \log f(\hat{\theta}|y) + \frac{1}{2}(\theta - \hat{\theta})' \left[\frac{d^2}{d\theta^2} \log f(\theta|y) \right]_{\theta=\hat{\theta}} (\theta - \hat{\theta})$$

$$I(\hat{\theta}) = \left[\frac{d^2}{d\theta^2} \log f(\theta|y) \right]_{\theta=\hat{\theta}}$$

Fisher information

Modal approximation

$$f(\theta|y) \approx f(\hat{\theta}|y) \exp\left[-\frac{1}{2}(\theta-\hat{\theta})' I(\hat{\theta})(\theta-\hat{\theta})\right]$$

$$P(\theta|y) \approx N(\hat{\theta}|y, I(\hat{\theta})^{-1})$$

weibull.py

```python
import pymc

# Some fake data
alpha = 3
beta = 5
N = 100
dataset = pymc.rweibull(alpha,beta, N)

# Model
a = pymc.Uniform('a', lower=0, upper=10, value=5, doc='Weibull alpha parameter')
b = pymc.Uniform('b', lower=0, upper=10, value=5, doc='Weibull beta parameter')
like = pymc.Weibull('like', alpha=a, beta=b, value=dataset, observed=True)
```

Line: 12 Column: 17 Python Soft Tabs: 4

```
>>> N = pymc.NormApprox(weibull)
>>> N.fit()
>>> N.mu[[N.a, N.b]]
array([ 3.22820192,  4.72739324])
>>> N.C[[N.a, N.b]]
matrix([[ 0.06548929,  0.01247454],
        [ 0.01247454,  0.02382098]])
```

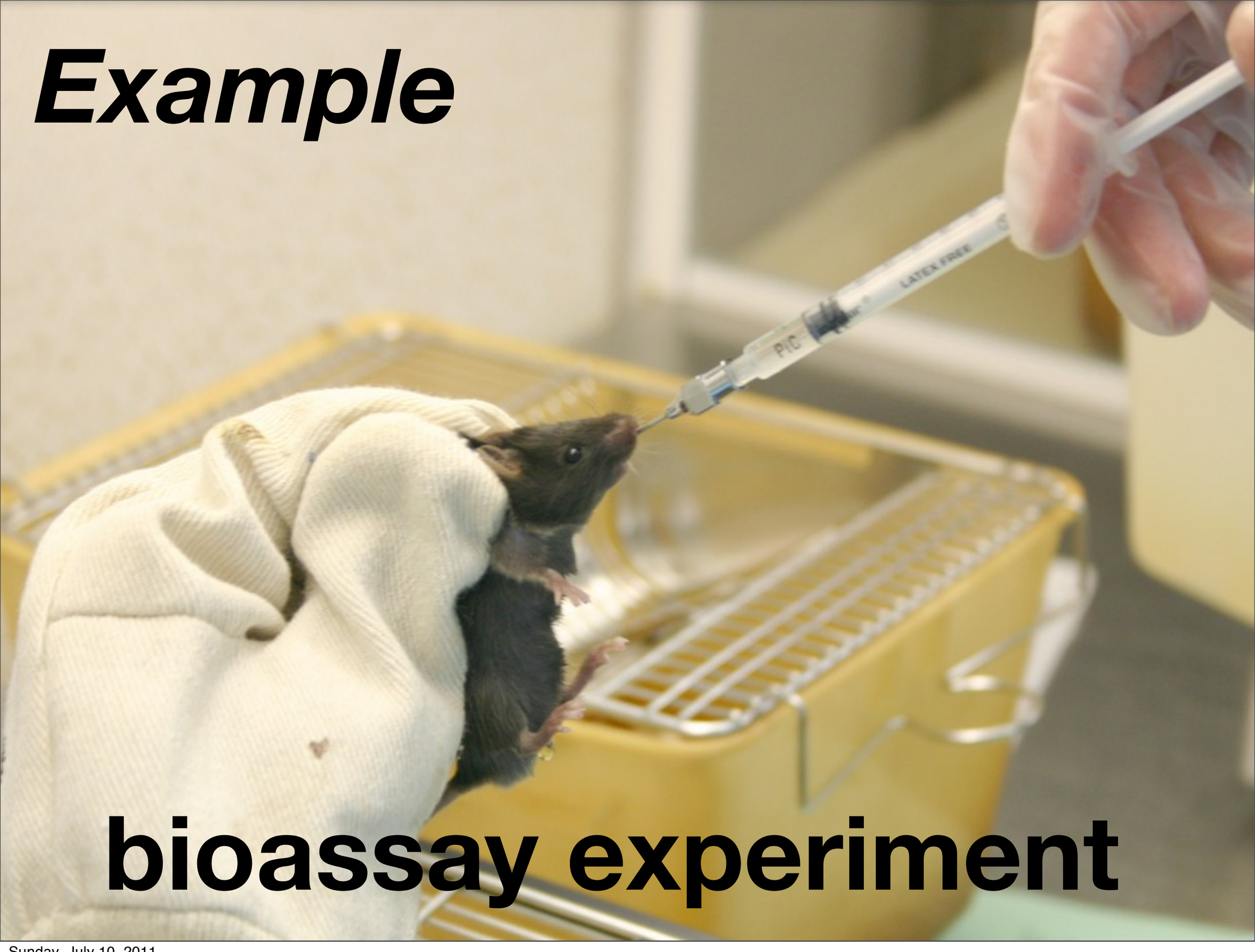
Extending PyMC

SciPy 2011

PyMC Objects

Node
Stochastic
Model
Sampler
StepMethod

Example



bioassay experiment

Dose (*log* g/ml)	Animals	Deaths
-0.863	5	0
-0.296	5	1
-0.053	5	3
0.727	5	5

bioassay data

$$y_i|\theta_i \sim \text{Bin}(n_i, \theta_i)$$

$$\text{logit}(\theta_i) = \alpha + \beta x_i$$

stochastic

bioassay.py

```
1  from pymc import *
2  from numpy import array
3
4  n = [5]*4
5  dose = [-.86,-.3,-.05,.73]
6  response = [0,1,3,5]
7
8  alpha = Normal('alpha', mu=0, tau=0.01)
9  beta = Normal('beta', mu=0, tau=0.01)
```

Line: 9 Column: 38 Python Soft Tabs: 4

deterministic

bioassay.py

```
1  from pymc import *
2  from numpy import array
3
4  n = [5]*4
5  dose = [-.86,-.3,-.05,.73]
6  response = [0,1,3,5]
7
8  alpha = Normal('alpha', mu=0, tau=0.01)
9  beta = Normal('beta', mu=0, tau=0.01)
10
11 @deterministic
12 def theta(a=alpha, b=beta):
13     """theta = invlogit(a + b*x)"""
14     return invlogit(a + b*array(dose))
```

Line: 14 Column: 39 | Python | Soft Tabs: 4 | theta(a=alpha, b=beta)

deterministic

bioassay.py

```
1  from pymc import *
2  from numpy import array
3
4  n = [5]*4
5  dose = [-.86,-.3,-.05,.73]
6  response = [0,1,3,5]
7
8  alpha = Normal('alpha', mu=0, tau=0.01)
9  beta = Normal('beta', mu=0, tau=0.01)
10
11 theta = Lambda('theta', lambda a=alpha, b=beta: invlogit(a + b*array(dose)))
```

Line: 11 Column: 77 Python Soft Tabs: 4

data stochastic

bioassay.py

```
1  from pymc import *
2  from numpy import array
3
4  n = [5]*4
5  dose = [-.86,-.3,-.05,.73]
6  response = [0,1,3,5]
7
8  alpha = Normal('alpha', mu=0, tau=0.01)
9  beta = Normal('beta', mu=0, tau=0.01)
10
11 theta = Lambda('theta', lambda a=alpha, b=beta: invlogit(a + b*array(dose)))
12
13 @observed
14 def deaths(value=response, n=n, p=theta):
15     """deaths ~ binomial_like(n, p)"""
16     return binomial_like_like(value, n, p)
```

Line: 16 Column: 43 | Python | Soft Tabs: 4 | deaths(value=response, n=n, p=theta)

data stochastic

bioassay.py

```
1  from pymc import *
2  from numpy import array
3
4  n = [5]*4
5  dose = [-.86,-.3,-.05,.73]
6  response = [0,1,3,5]
7
8  alpha = Normal('alpha', mu=0, tau=0.01)
9  beta = Normal('beta', mu=0, tau=0.01)
10
11 theta = Lambda('theta', lambda a=alpha, b=beta: invlogit(a + b*array(dose)))
12
13 deaths = Binomial('deaths', n=n, p=theta, value=response, observed=True)
```

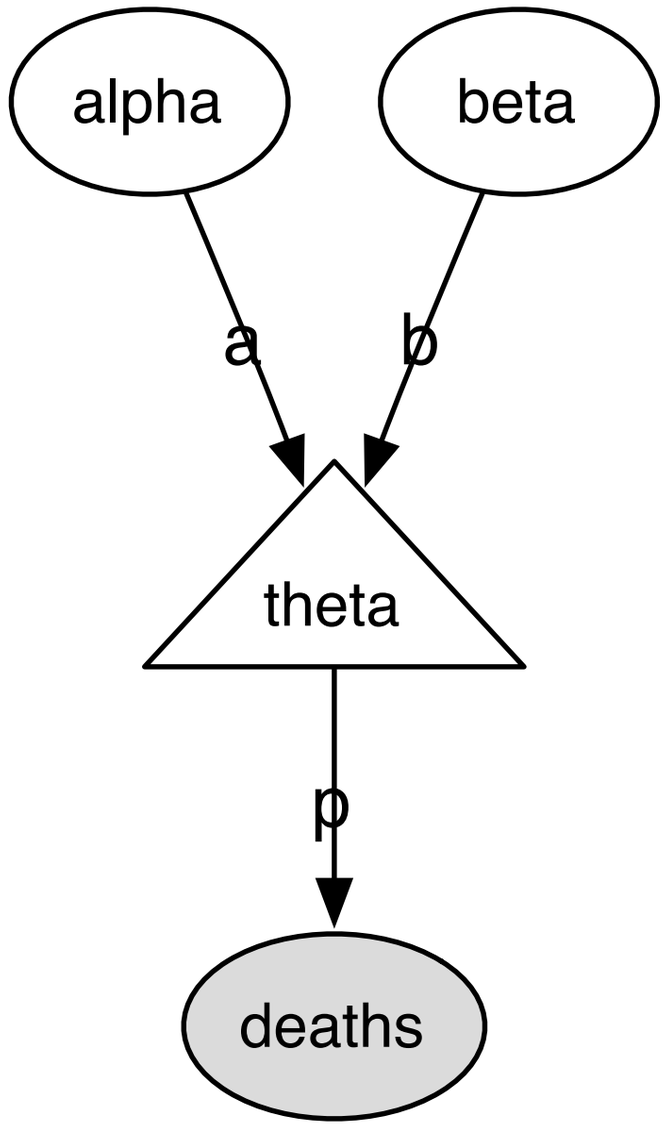
Line: 13 Column: 73 Python Soft Tabs: 4

data stochastic

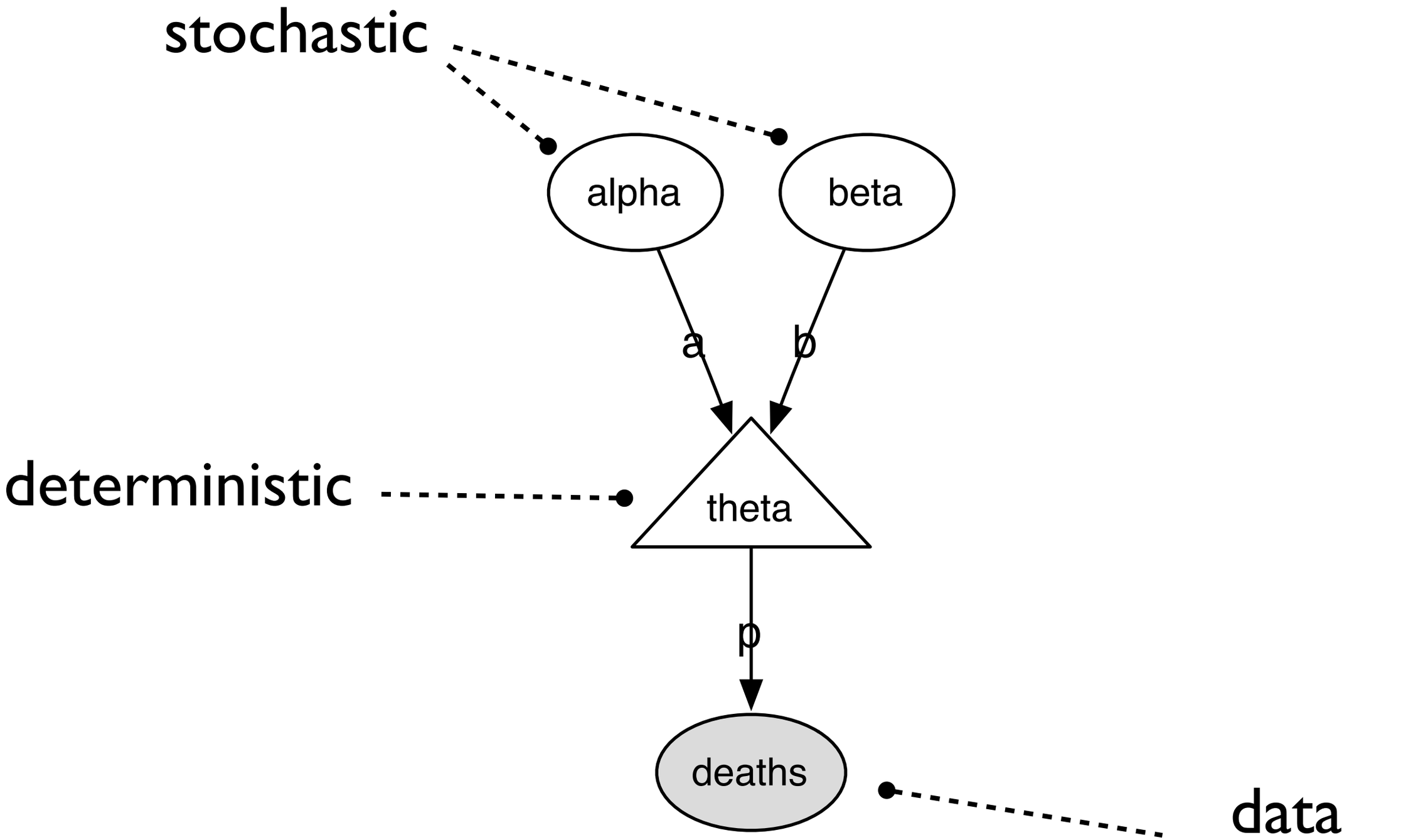
bioassay.py

```
1  from pymc import *
2  from numpy import array
3
4  n = [5]*4
5  dose = [-.86,-.3,-.05,.73]
6  response = [0,1,3,5]
7
8  alpha = Normal('alpha', mu=0, tau=0.01)
9  beta = Normal('beta', mu=0, tau=0.01)
10
11 theta = Lambda('theta', lambda a=alpha, b=beta: invlogit(a + b*array(dose)))
12
13 deaths = Binomial('deaths', n=n, p=theta, value=response, observed=True)
```

Line: 13 Column: 73 Python Soft Tabs: 4




```
>>> graph.dag(M)
```


stochastic
alpha
beta
a
b
deterministic
theta
p
deaths
data
stochastic

>>> graph.dag(M)

factor potential

$$p(\theta, \phi | y) \quad \propto \quad p(y|\theta) p(\theta|\phi) p(\phi)$$

$$\Downarrow$$

$$p(\theta, \phi, \tau | y) \quad \propto \quad p(y|\theta) p(\theta|\phi) p(\theta|\tau) p(\tau) p(\phi)$$

arbitrary log-probability terms

factor potential

example.py

```
1  @potential
2  def mean_zero(y=theta, mu=0, tau=tau):
3      """mean_zero = normal(y, mu, tau)"""
4      return normal_like(y, mu, tau)
5
```

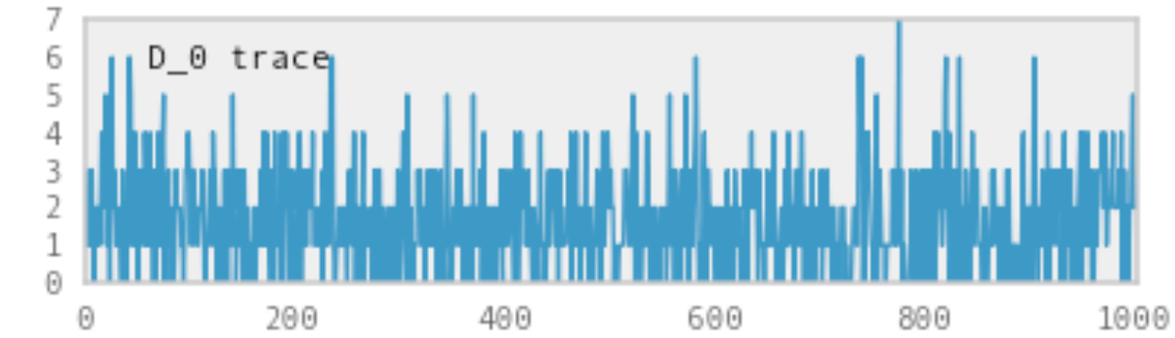
Line: 4 Column: 23 Python Soft Tabs: 4 m.

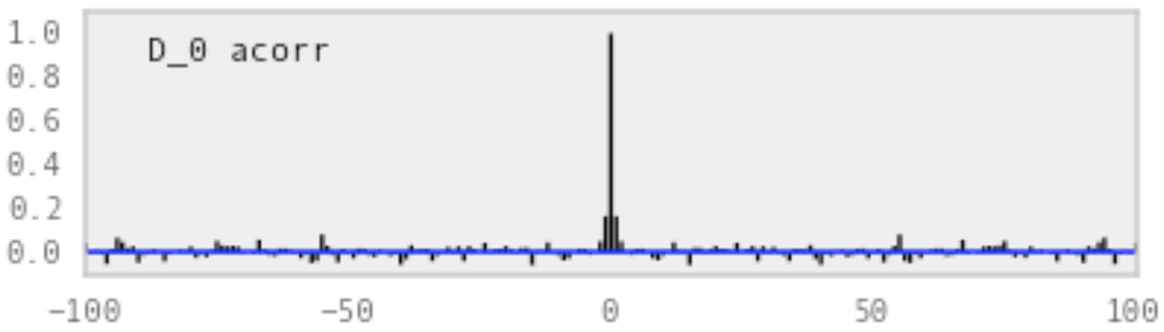
missing data

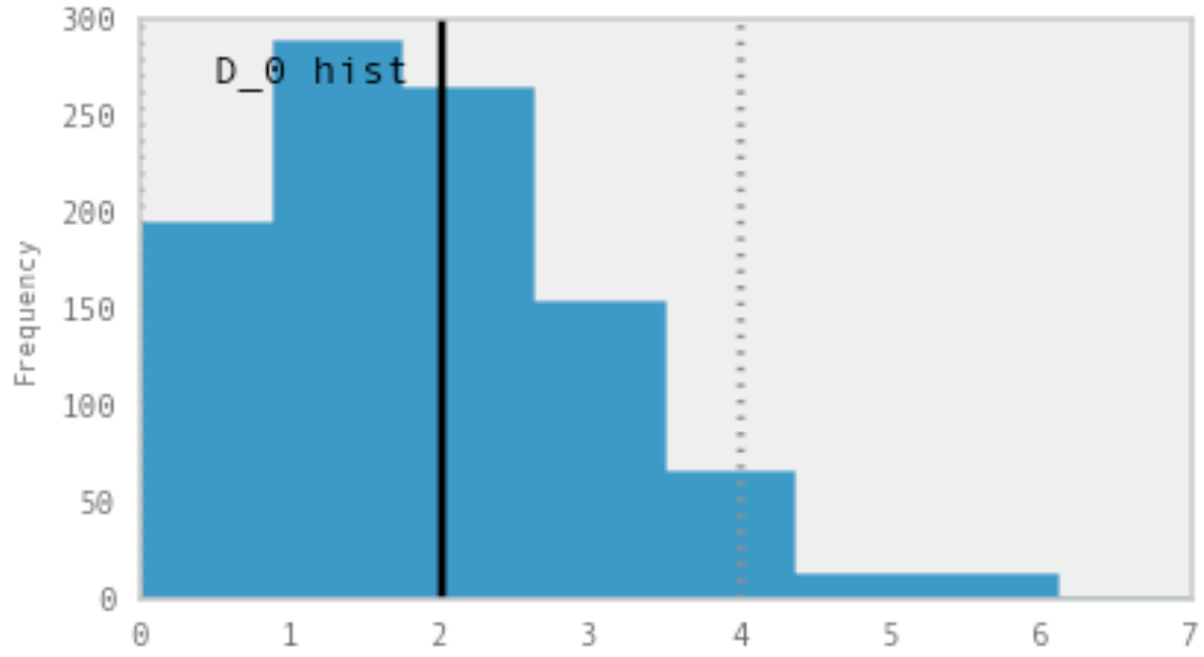
missing_count.py

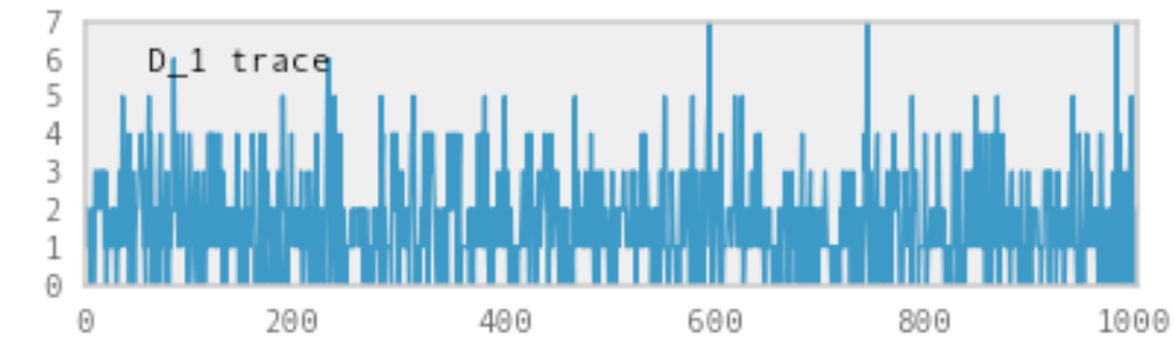
```
1  import numpy as np
2  from numpy.ma import masked_array
3  from pymc import *
4
5  # Missing values indicated by negative-valued placeholders
6  count =   np.array([ 4, 5, 4, 0, 1, 4, 3, 4, 0, 6, 3, 3, 4, 0, 2, 6,
7  3, 3, 5, 4, 5, 3, 1, 4, 4, 1, 5, 5, 3, 4, 2, 5,
8  2, 2, 3, 4, 2, 1, 3, -999, 2, 1, 1, 1, 1, 3, 0, 0,
9  1, 0, 1, 1, 0, 0, 3, 1, 0, 3, 2, 2, 0, 1, 1, 1,
10 0, 1, 0, 1, 0, 0, 0, 2, 1, 0, 0, 0, 1, 1, 0, 2,
11 3, 3, 1, -999, 2, 1, 1, 1, 1, 2, 4, 2, 0, 0, 1, 4,
12 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1])
13
14 # Mean count
15 m = Exponential('mu', beta=1)
16
17 # Create masked array with data
18 masked_values = masked_array(count, mask=count==-999)
19
20 # Pass masked array to data stochastic, and it does the right thing
21 D = Poisson('D', mu=m, value=masked_values, observed=True)
```

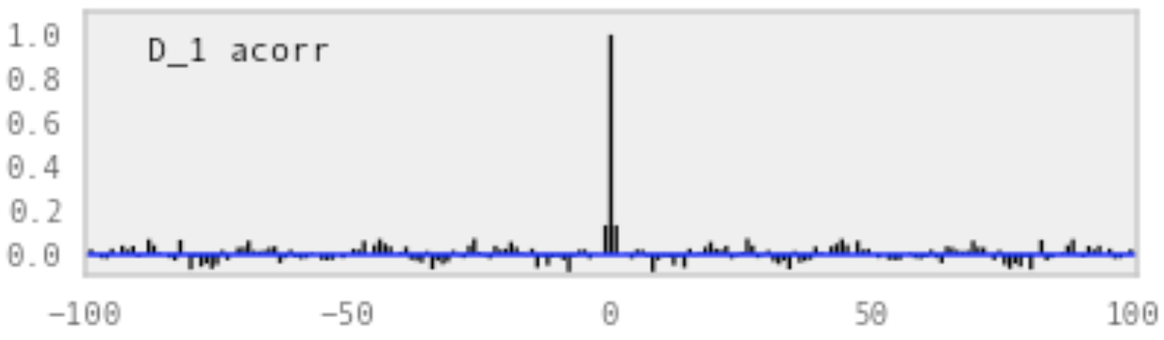
Line: 18 Column: 1 Python Soft Tabs: 4 —

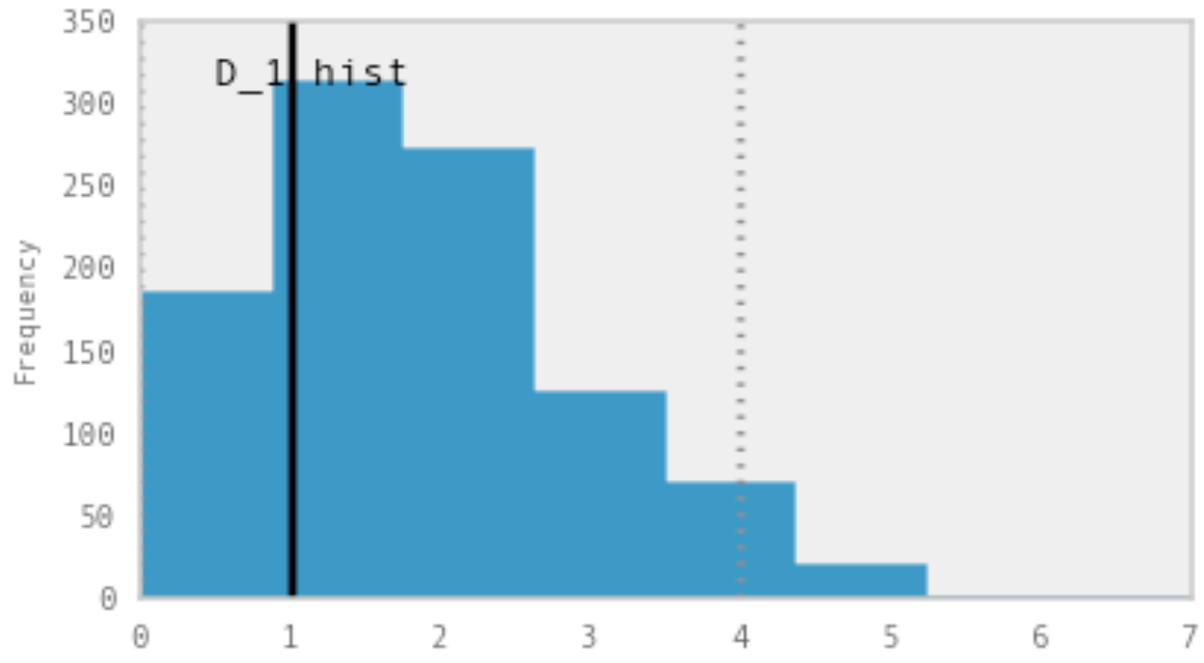

D_0 trace
0
1
2
3
4
5
6
7
0
200
400
600
800
1000

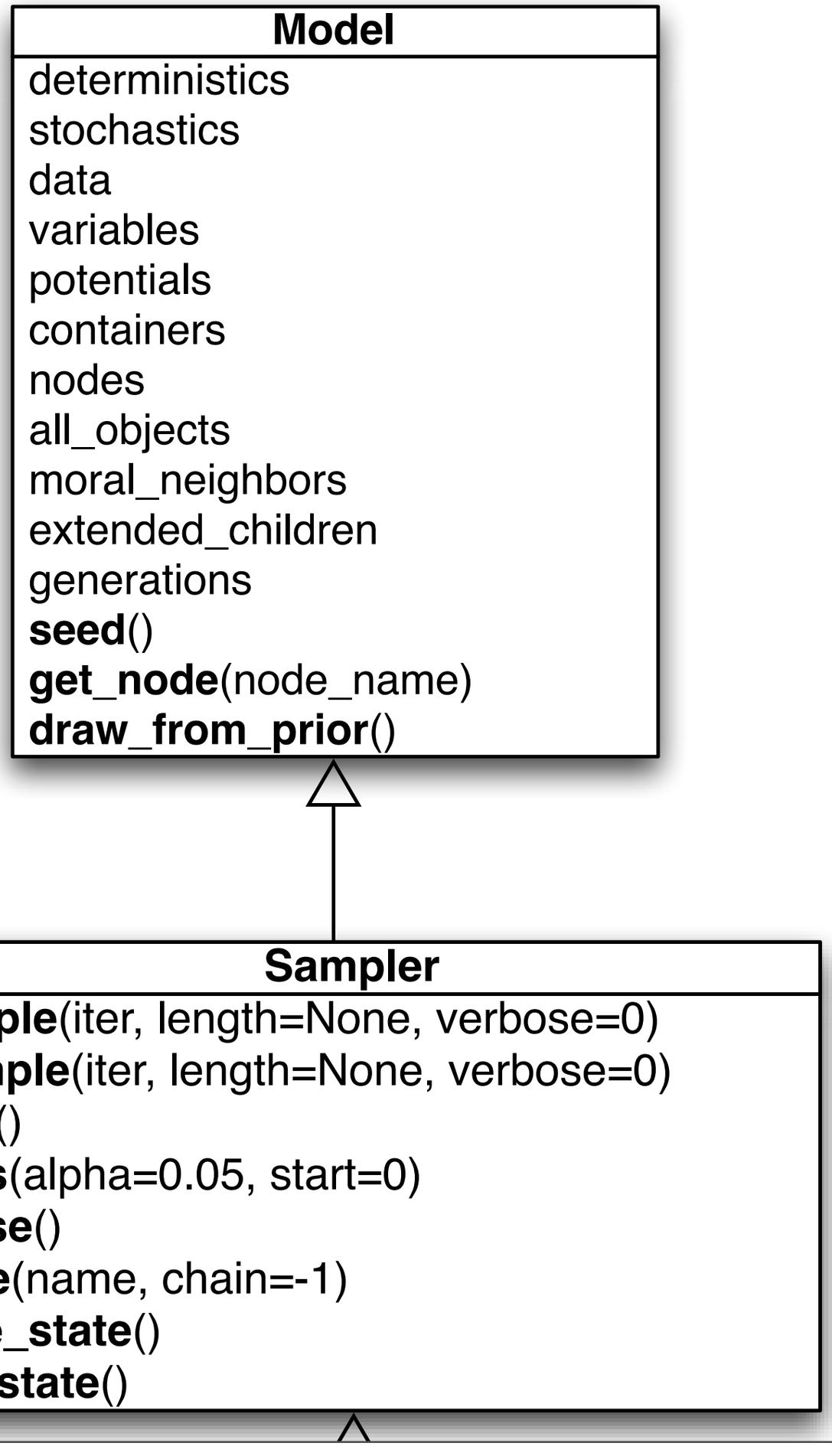

D_0 acorr
0.0
0.2
0.4
0.6
0.8
1.0
−100
−50
0
50
100

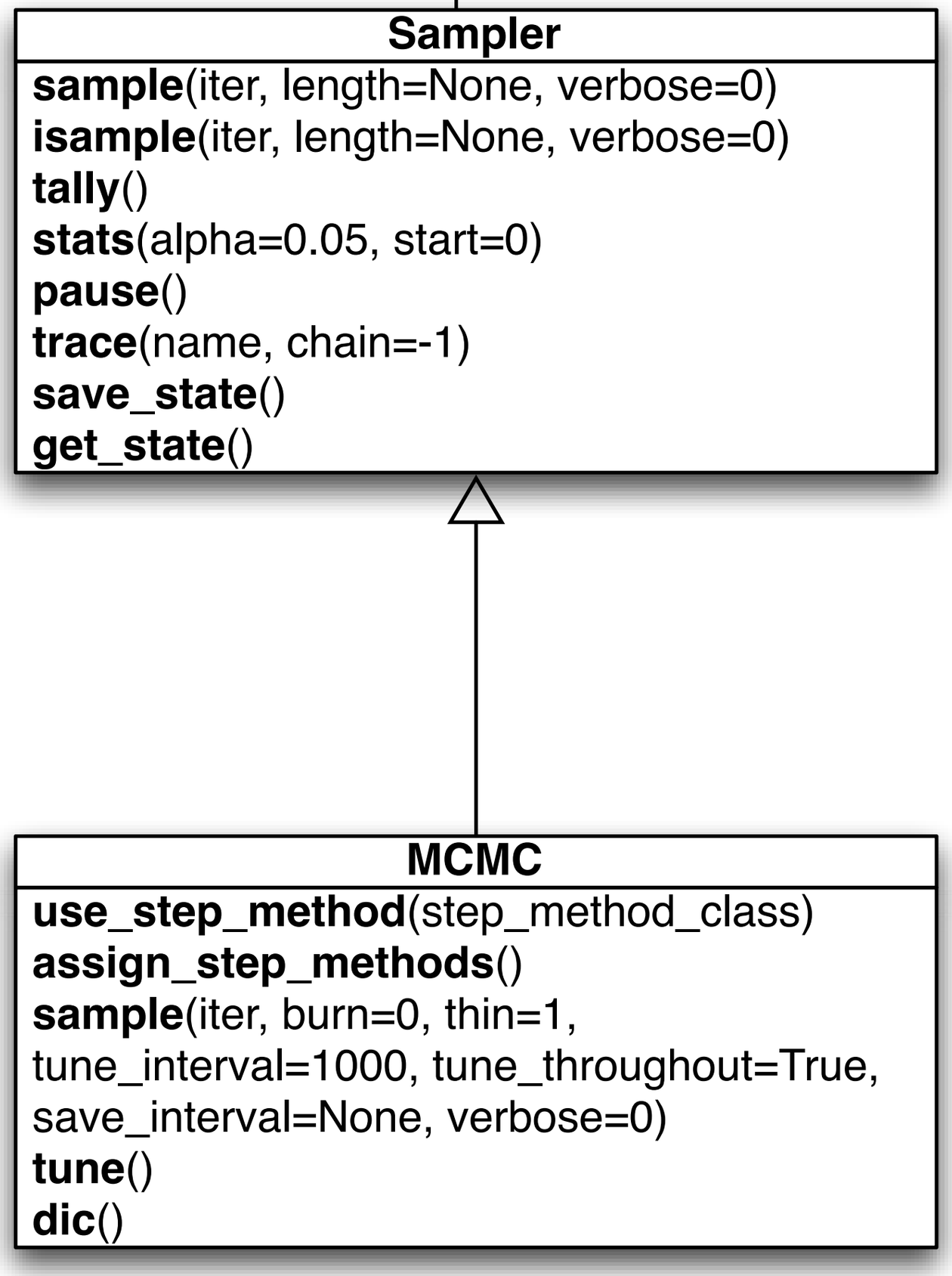

D_0 hist
Frequency
0
50
100
150
200
250
300
0
1
2
3
4
5
6
7


D_1 trace
0
1
2
3
4
5
6
7
0
200
400
600
800
1000

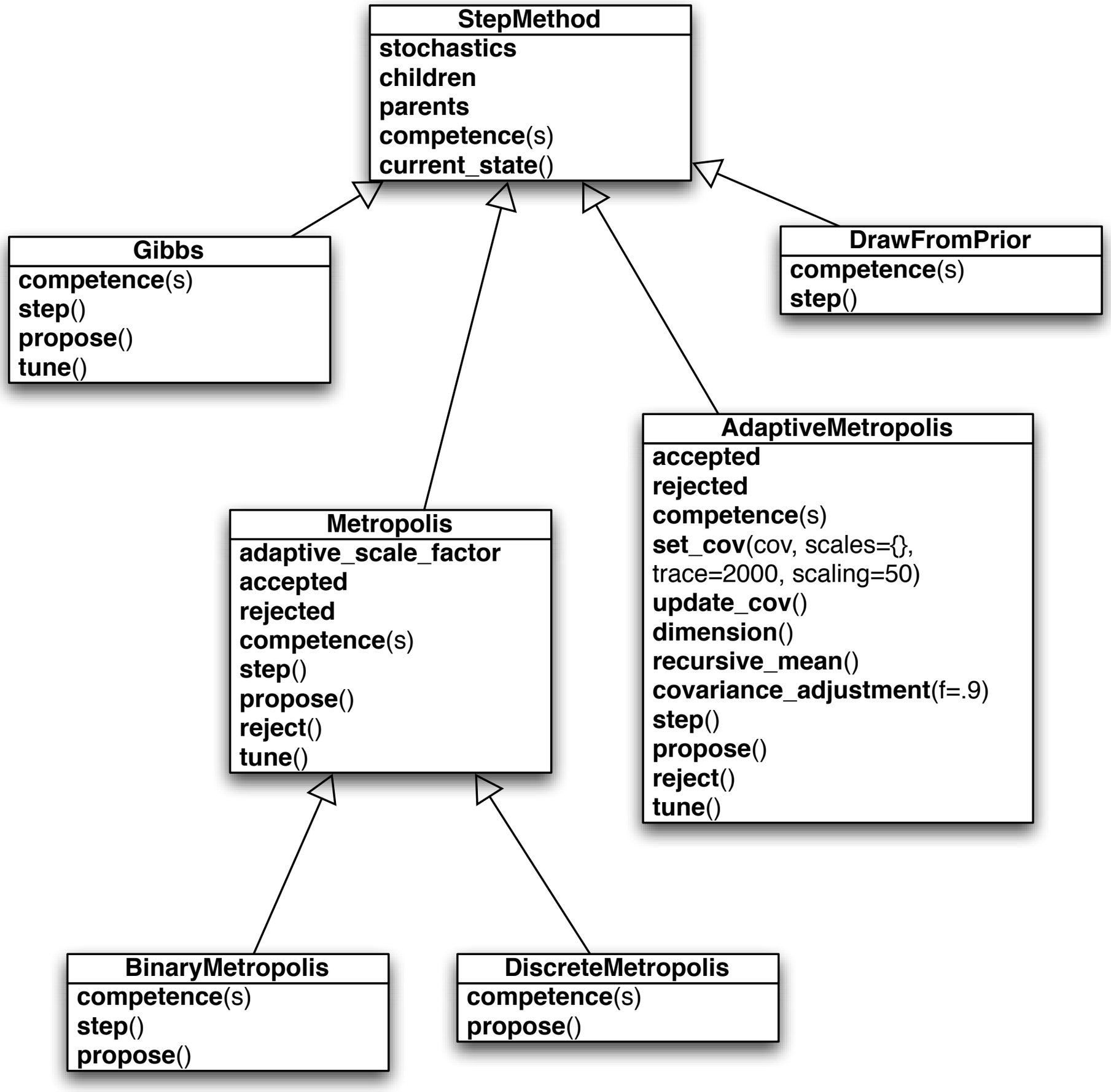

D_1 acorr
0.0
0.2
0.4
0.6
0.8
1.0
−100
−50
0
50
100


D_1 hist
Frequency
0
50
100
150
200
250
300
350
0
1
2
3
4
5
6
7


Model
deterministics
stochastics
data
variables
potentials
containers
nodes
all_objects
moral_neighbors
extended_children
generations
seed()
get_node(node_name)
draw_from_prior()
Sampler
sample(iter, length=None, verbose=0)
isample(iter, length=None, verbose=0)
tally()
stats(alpha=0.05, start=0)
pause()
trace(name, chain=-1)
save_state()
get_state()


Sampler
sample(iter, length=None, verbose=0)
isample(iter, length=None, verbose=0)
tally()
stats(alpha=0.05, start=0)
pause()
trace(name, chain=-1)
save_state()
get_state()
MCMC
use_step_method(step_method_class)
assign_step_methods()
sample(iter, burn=0, thin=1,
tune_interval=1000, tune_throughout=True,
save_interval=None, verbose=0)
tune()
dic()

Step Methods


StepMethod
stochastics
children
parents
competence(s)
current_state()
Gibbs
competence(s)
step()
propose()
tune()
DrawFromPrior
competence(s)
step()
Metropolis
adaptive_scale_factor
accepted
rejected
competence(s)
step()
propose()
reject()
tune()
AdaptiveMetropolis
accepted
rejected
competence(s)
set_cov(cov, scales={},
trace=2000, scaling=50)
update_cov()
dimension()
recursive_mean()
covariance_adjustment(f=.9)
step()
propose()
reject()
tune()
BinaryMetropolis
competence(s)
step()
propose()
DiscreteMetropolis
competence(s)
propose()

StepMethod methods

StepMethods.py

```
422
423     def step(self):
424         """
425         The default step method applies if the variable is floating-point
426         valued, and is not being proposed from its prior.
427         """
428
429         # Probability and likelihood for s's current value:
430
431         if self.verbose>2:
432             print
433             print self._id + ' getting initial logp.'
434
435         if self.proposal_distribution == "Prior":
436             logp = self.loglike
437         else:
438             logp = self.logp_plus_loglike
439
440         if self.verbose>2:
441             print self._id + ' proposing.'
442
443         # Sample a candidate value
444         self.propose()
445
446         # Probability and likelihood for s's proposed value:
447         try:
448             if self.proposal_distribution == "Prior":
449                 logp_p = self.loglike
450                 # Check for weirdness before accepting jump
451                 if self.check_before_accepting:
452                     self.stochastic.logp
453             else:
454                 logp_p = self.logp_plus_loglike
455
456         except ZeroProbability:
457
458             # Reject proposal
```

```
    def step(self):
        """
        The default step method applies if the variable is floating-point
        valued, and is not being proposed from its prior.
        """

        # Probability and likelihood for s's current value:

        if self.verbose>2:
            print
            print self._id + ' getting initial logp.'

        if self.proposal_distribution == "Prior":
            logp = self.loglike
        else:
            logp = self.logp_plus_loglike

        if self.verbose>2:
            print self._id + ' proposing.'

        # Sample a candidate value
        self.propose()

        # Probability and likelihood for s's proposed value:
        try:
            if self.proposal_distribution == "Prior":
                logp_p = self.loglike
                # Check for weirdness before accepting jump
                if self.check_before_accepting:
                    self.stochastic.logp
            else:
                logp_p = self.logp_plus_loglike

        except ZeroProbability:
```

StepMethod methods

StepMethods.py

```
514
515     def tune(self, divergence_threshold=1e10, verbose=0):
516         """
517         Tunes the scaling parameter for the proposal distribution
518         according to the acceptance rate of the last k proposals:
519
520         Rate    Variance adaptation
521         ----    -------------------
522         <0.001        x 0.1
523         <0.05         x 0.5
524         <0.2          x 0.9
525         >0.5          x 1.1
526         >0.75         x 2
527         >0.95         x 10
528
529         This method is called exclusively during the burn-in period of the
530         sampling algorithm.
531
532         May be overridden in subclasses.
533         """
534
535         if self.verbose is not None:
536             verbose = self.verbose
537
538         if self.verbose is not None:
539             verbose = self.verbose
540
541         # Verbose feedback
542         if verbose > 0:
543             print '\t%s tuning:' % self._id
544
545         # Flag for tuning state
546         tuning = True
547
548         # Calculate recent acceptance rate
549         if not (self.accepted + self.rejected): return tuning
550         acc_rate = self.accepted / (self.accepted + self.rejected)
551
```

```
514
515     def tune(self, divergence_threshold=1e10, verbose=0):
516         """
517         Tunes the scaling parameter for the proposal distribution
518         according to the acceptance rate of the last k proposals:
519
520         Rate    Variance adaptation
521         ----    -------------------
522         <0.001        x 0.1
523         <0.05         x 0.5
524         <0.2          x 0.9
525         >0.5          x 1.1
526         >0.75         x 2
527         >0.95         x 10
528
529         This method is called exclusively during the burn-in period of the
530         sampling algorithm.
531
532         May be overridden in subclasses.
533         """
534
535         if self.verbose is not None:
536             verbose = self.verbose
537
538         if self.verbose is not None:
539             verbose = self.verbose
540
541         # Verbose feedback
542         if verbose > 0:
543             print '\t%s tuning:' % self._id
544
545         # Flag for tuning state
546         tuning = True
547
548         # Calculate recent acceptance rate
```

StepMethod methods

StepMethods.py

```
        else:
            if proposal_distribution.capitalize() in self._valid_proposals:
                self.proposal_distribution = proposal_distribution
            else:
                raise ValueError, "Invalid proposal distribution '%s'
specified for Metropolis sampler." % proposal_distribution

    _valid_proposals = ['Normal', 'Prior']

    @staticmethod
    def competence(s):
        """
        The competence function for Metropolis
        """
        if s.dtype is None:
            return .5

        if not s.dtype in float_dtypes:
            # If the stochastic's binary or discrete, I can't do it.
            return 0
        else:
            return 1

    def hastings_factor(self):
        """
        If this is a Metropolis-Hastings method (proposal is not symmetric
random walk),
        this method should return log(back_proposal) - log(forward_proposal).
        """
        return 0.

    def step(self):
        """
        The default step method applies if the variable is floating-point
        valued, and is not being proposed from its prior.
        """

        # Probability and likelihood for s's current value:
```

```
401
402     @staticmethod
403     def competence(s):
404         """
405         The competence function for Metropolis
406         """
407         if s.dtype is None:
408             return .5
409
410         if not s.dtype in float_dtypes:
411             # If the stochastic's binary or discrete, I can't do it.
412             return 0
413         else:
414             return 1
415
416     def hastings_factor(self):
417         """
418         If this is a Metropolis-Hastings method (proposal is not symmetric
    random walk),
419         this method should return log(back_proposal) - log(forward_proposal).
420         """
421         return 0.
422
423     def step(self):
424         """
425         The default step method applies if the variable is floating-point
426         valued, and is not being proposed from its prior.
427         """
428
429         # Probability and likelihood for s's current value:
430
431         if self.verbose>2:
432             print
433             print self._id + ' getting initial logp.'
434
435         if self.proposal_distribution == "Prior":
```

Competence

0 = I cannot safely handle this variable.
1 = I can handle the variable about as well as the standard Metropolis step method.
2 = I can do better than Metropolis.
3 = I am the best step method you are likely to find for this variable in most cases.

```
S.use_step_method(better_method,
                  my_stochastic,
                  *args,
                  **kwargs)
```

```
>>> mu = pymc.Normal('mu', 0, .01,
value=0)

>>> tau = pymc.Exponential('tau', .01,
value=1)

>>> cutoff = pymc.Exponential('cutoff', 1,
value=1.3)

>>> D = pymc.TruncatedNormal('D', mu, tau,
-numpy.inf, cutoff, value=data,
observed=True)
```

$$q(x', x^{(j)}) = \text{TruncatedNormal}(x, \sigma, \max(D), \infty)$$

truncated normal proposal

truncated_metropolis.py

```
class TruncatedMetropolis(pymc.Metropolis):
    def __init__(self, stochastic, low_bound, up_bound, *args, **kwargs):
        self.low_bound = low_bound
        self.up_bound = up_bound
        pymc.Metropolis.__init__(self, stochastic, *args, **kwargs)

    # Propose method generates proposal values
    def propose(self):
        tau = 1./(self.adaptive_scale_factor * self.proposal_sd)**2
        self.stochastic.value = \
            pymc.rtruncnorm(self.stochastic.value, tau, self.low_bound,
self.up_bound)

    # Hastings factor method accounts for asymmetric proposal distribution
    def hastings_factor(self):
        tau = 1./(self.adaptive_scale_factor * self.proposal_sd)**2
        cur_val = self.stochastic.value
        last_val = self.stochastic.last_value

        lp_for = pymc.truncnorm_like(cur_val, last_val, tau, \
            self.low_bound, self.up_bound)
        lp_bak = pymc.truncnorm_like(last_val, cur_val, tau, \
            self.low_bound, self.up_bound)

        if self.verbose > 1:
            print self._id + ': Hastings factor %f'%(lp_bak - lp_for)
        return lp_bak - lp_for
```

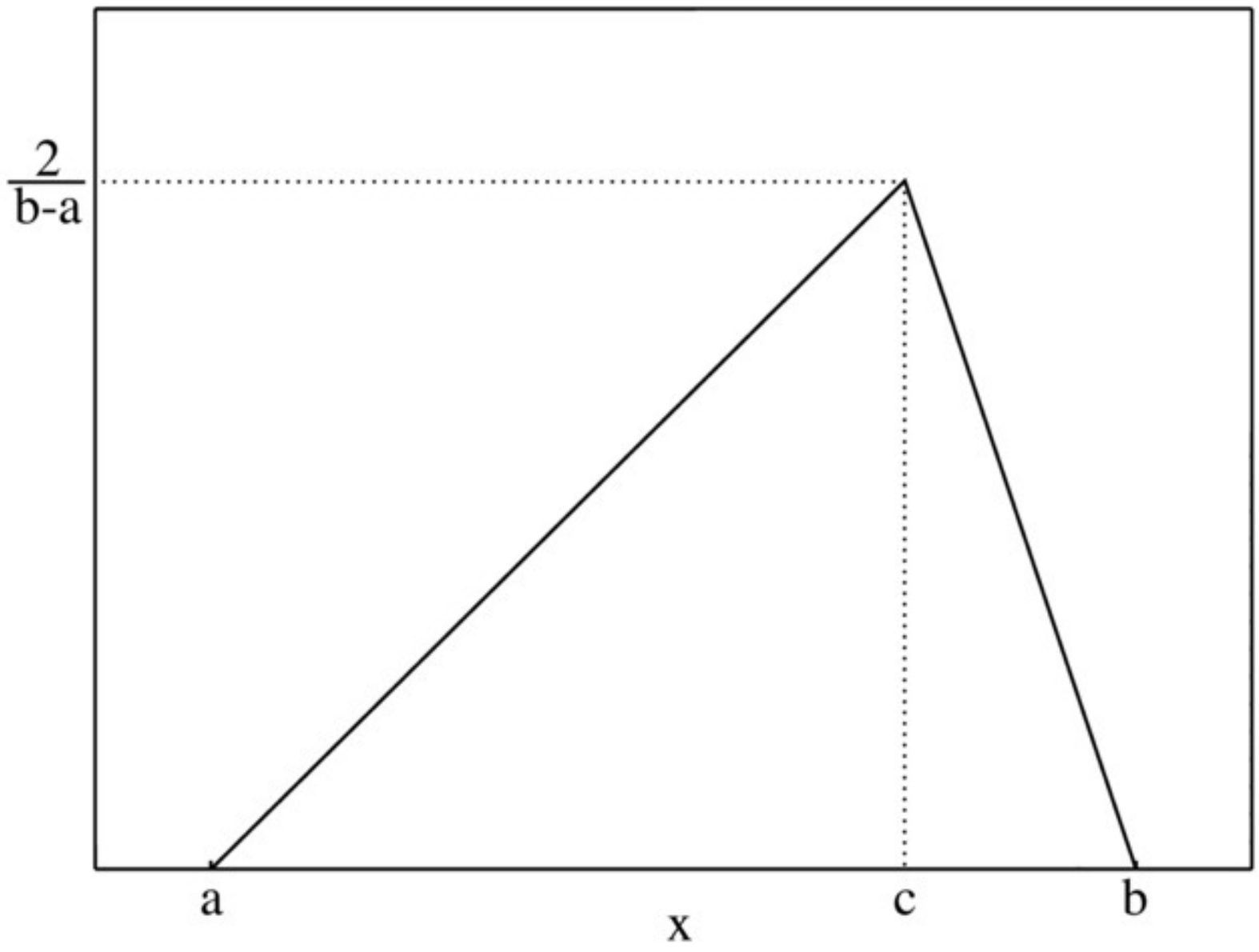
Line: 26 Column: 31 | Python | Soft Tabs: 4 | hastings_factor(self)

Custom Stochastics

Stochastic attributes

(1) name
(2) `logp` method
(3) `random` method
(4) dtype
(5) `mv` flag

example



triangle distribution

triangle distribution

$$f(x) = \begin{cases} \dfrac{2(x - \min)}{(\text{mode} - \min)(\max - \min)} & \text{if } \min \leq x \leq \text{mode} \\ \dfrac{2(\max - x)}{(\max - \min)(\max - \text{mode})} & \text{if mode} \leq x \leq \max \end{cases}$$

distribution function

triangular.py

```python
from numpy import log, random, sqrt, zeros, atleast_1d

def triangular_like(x, mode, minval, maxval):
    """Log-likelihood of triangular distribution"""

    x = atleast_1d(x)

    # Check for support
    if any(x<minval) or any(x>maxval): return -inf

    # Likelihood of left values
    like = sum(log(2*(x[x<=mode] - minval)) - log(mode - minval) - log(maxval
- minval))

    # Likelihood of right values
    like += sum(log(2*(maxval - x[x>mode])) - log(maxval - minval) -
log(maxval - mode))

    return like

def rtriangular(mode, minval, maxval, size=1):
    """Generate triangular random numbers"""

    # Uniform random numbers
    z = atleast_1d(random.random(size))

    # Threshold for transformation
    threshold = (mode - minval)/(maxval - minval)

    # Transform uniforms
    u = atleast_1d(zeros(size))
    u[z<=threshold] = minval + sqrt(z[z<=threshold]*(maxval - minval)*(mode -
minval))
    u[z>threshold] = maxval - sqrt((1 - z[z>threshold])*(maxval -
minval)*(maxval - mode))

    return u
```

Line: 6 Column: 21 | Python | Soft Tabs: 4 | triangular_like(x, mode, minval, maxval)

triangular.py

```
from numpy import log, random, sqrt, zeros, atleast_1d

def triangular_like(x, mode, minval, maxval):
    """Log-likelihood of triangular distribution"""

    x = atleast_1d(x)

    # Check for support
    if any(x<minval) or any(x>maxval): return -inf

    # Likelihood of left values
    like = sum(log(2*(x[x<=mode] - minval)) - log(mode - minval) - log(maxval
- minval))

    # Likelihood of right values
    like += sum(log(2*(maxval - x[x>mode])) - log(maxval - minval) -
log(maxval - mode))

    return like

def rtriangular(mode, minval, maxval, size=1):
    """Generate triangular random numbers"""

    # Uniform random numbers
    z = atleast_1d(random.random(size))
```

```
    # Likelihood of right values
    like += sum(log(2*(maxval - x[x>mode])) - log(maxval - minval) -
log(maxval - mode))

    return like

def rtriangular(mode, minval, maxval, size=1):
    """Generate triangular random numbers"""

    # Uniform random numbers
    z = atleast_1d(random.random(size))

    # Threshold for transformation
    threshold = (mode - minval)/(maxval - minval)

    # Transform uniforms
    u = atleast_1d(zeros(size))
    u[z<=threshold] = minval + sqrt(z[z<=threshold]*(maxval - minval)*(mode -
minval))
    u[z>threshold] = maxval - sqrt((1 - z[z>threshold])*(maxval -
minval)*(maxval - mode))

    return u
```

6 Column: 21 | Python | Soft Tabs: 4 | triangular_like(x, mode, minval, maxval

```
>>> Triangular = pymc.stochastic_from_dist
('Triangular', logp=triangular_like,
random=rtriangular, dtype=float)
```

```
>>> x = Triangular('x', mode=3, minval=1,
maxval=10)

>>> x.value
array(6.0565769362029522)

>>> x.logp
-2.0779384017901741

>>> x.random()
array(5.2428089474233568)
```

Gaussian Processes

SciPy 2011

Having observed data
{X,Y}, we want to find
suitable function **f**

$$Y = f(X) + \epsilon$$

Gaussian process

“a collection of random variables, any finite number of which have a joint Gaussian distribution”

“a distribution over functions”

Gaussian process

$$f \sim \mathcal{GP}(m(\mathbf{x}), C(\mathbf{x}, \mathbf{x}))$$

Gaussian process

covariance function

$$f \sim \mathcal{GP}(m(\mathbf{x}), C(\mathbf{x}, \mathbf{x}))$$

mean function

$$P(f(\mathbf{x}))|y_N, X_N) = \frac{P(y_N|f(\mathbf{x}), X_N)P(f(\mathbf{x}))}{P(y_N|X_N)}$$

GPP

$$P(f(\mathbf{x}))|y_N, X_N) = \frac{P(y_N|f(\mathbf{x}), X_N)P(f(\mathbf{x}))}{P(y_N|X_N)}$$

Mean function

gpexample.py

```
1  from pymc.gp import *
2
3  # Generate mean
4  def quadfun(x, a, b, c):
5      return (a * x ** 2 + b * x + c)
6
7  M = Mean(quadfun, a = 1., b = .5, c = 2.)
```

Line: 7 Column: 42 Python Soft Tabs: 4 quadfun(x, a, b, c)

Covariance Function

$$C(d) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\, d}{l} \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu}\, d}{l} \right)$$

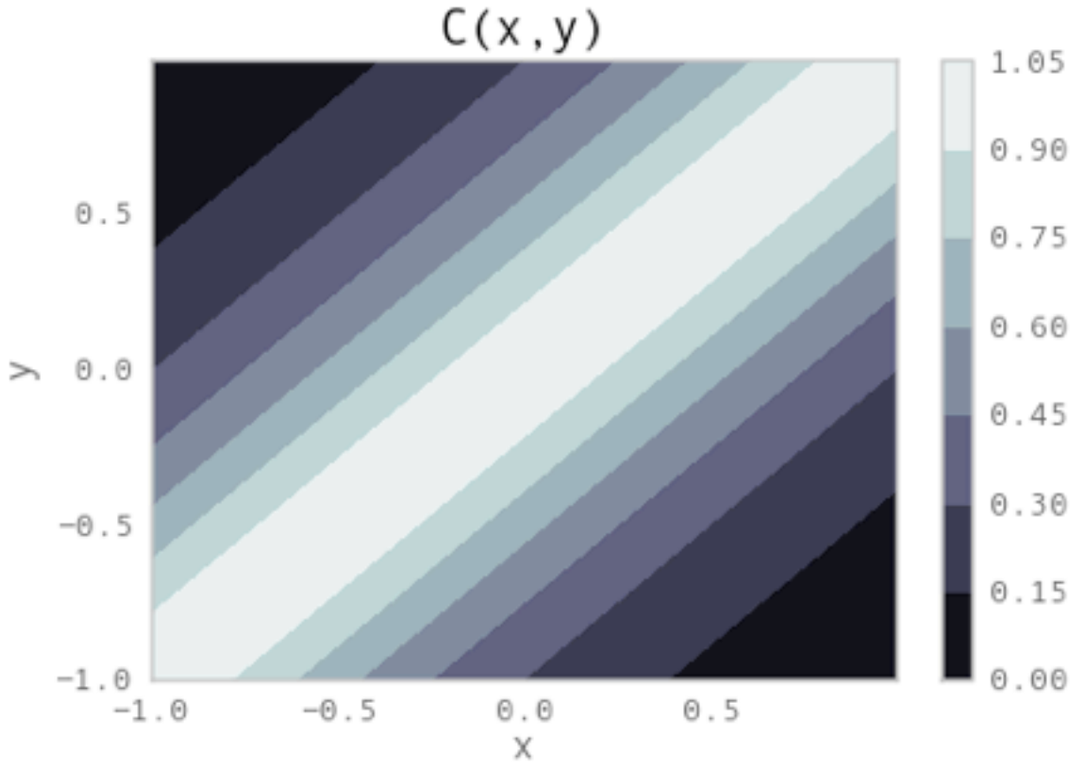
Matérn covariance

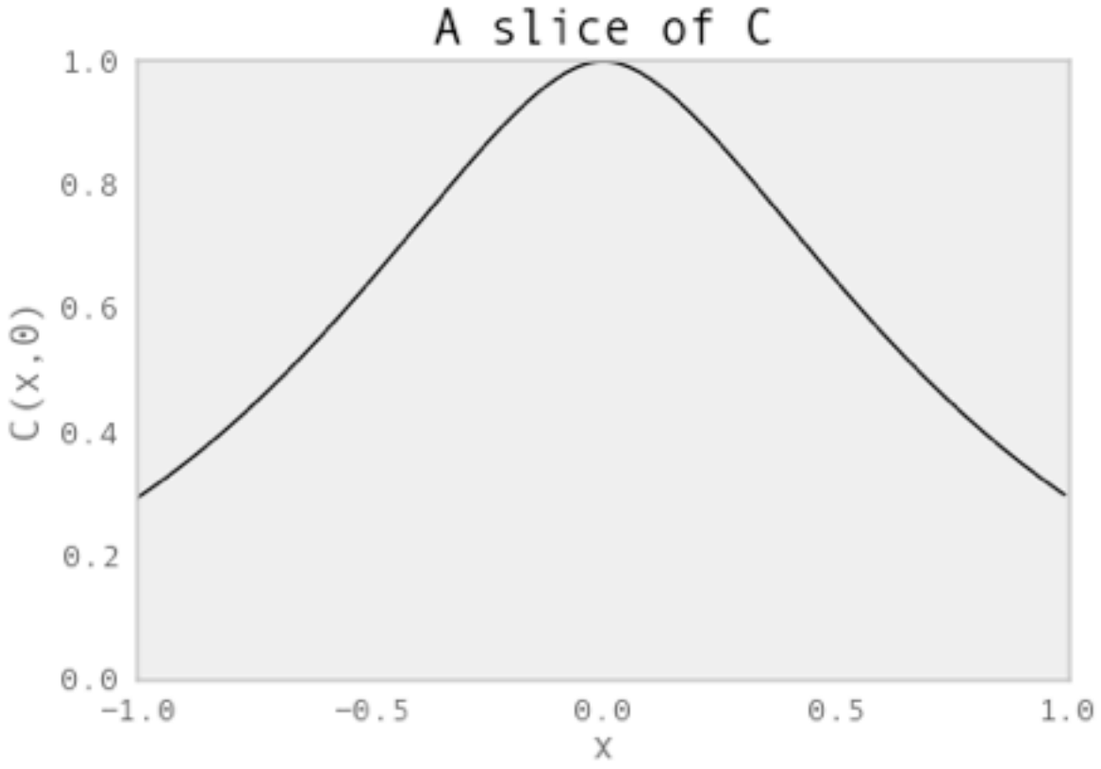
Covariance Function

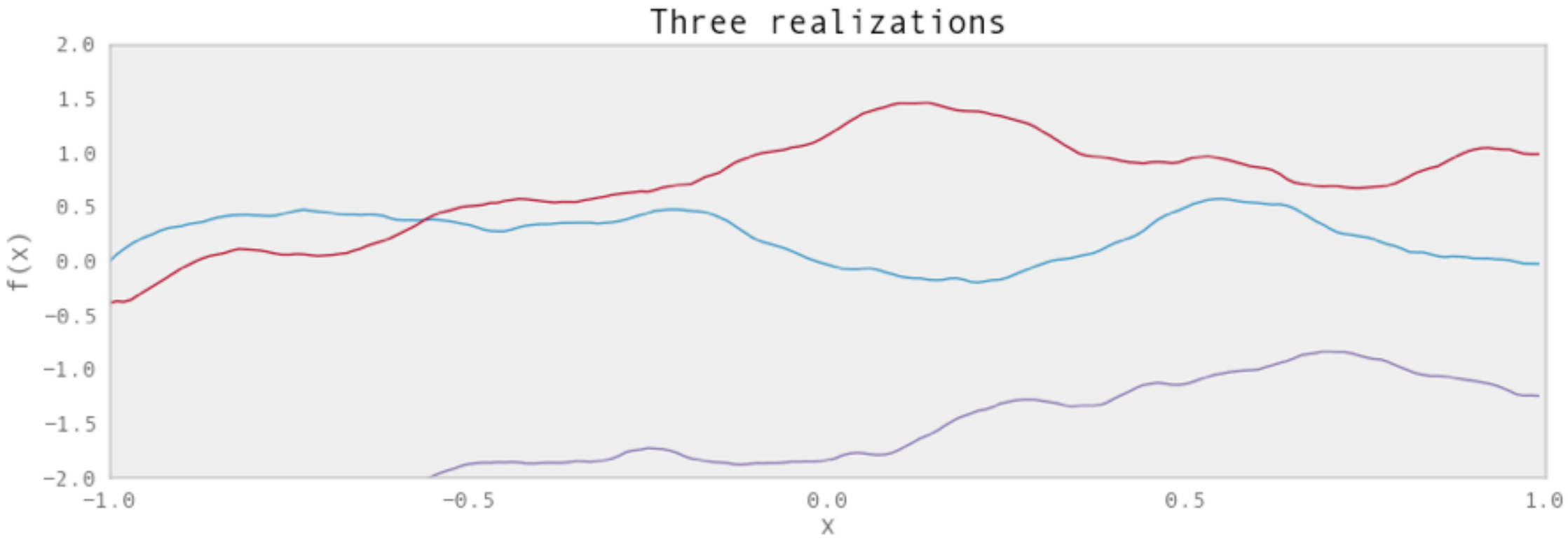
gpexample.py

```
1   from pymc.gp import *
2
3   # Generate mean
4   def quadfun(x, a, b, c):
5       return (a * x ** 2 + b * x + c)
6
7   M = Mean(quadfun, a = 1., b = .5, c = 2.)
8
9   C = Covariance(eval_fun = matern.euclidean, diff_degree = 1.4, amp = .4, scale
    = 1.)
10
```

Line: 9 Column: 1 | Python | Soft Tabs: 4 | quadfun(x, a, b, c)


C(x,y)
y
x
1.05
0.90
0.75
0.60
0.45
0.30
0.15
0.00
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5


A slice of C
C(x,0)
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0

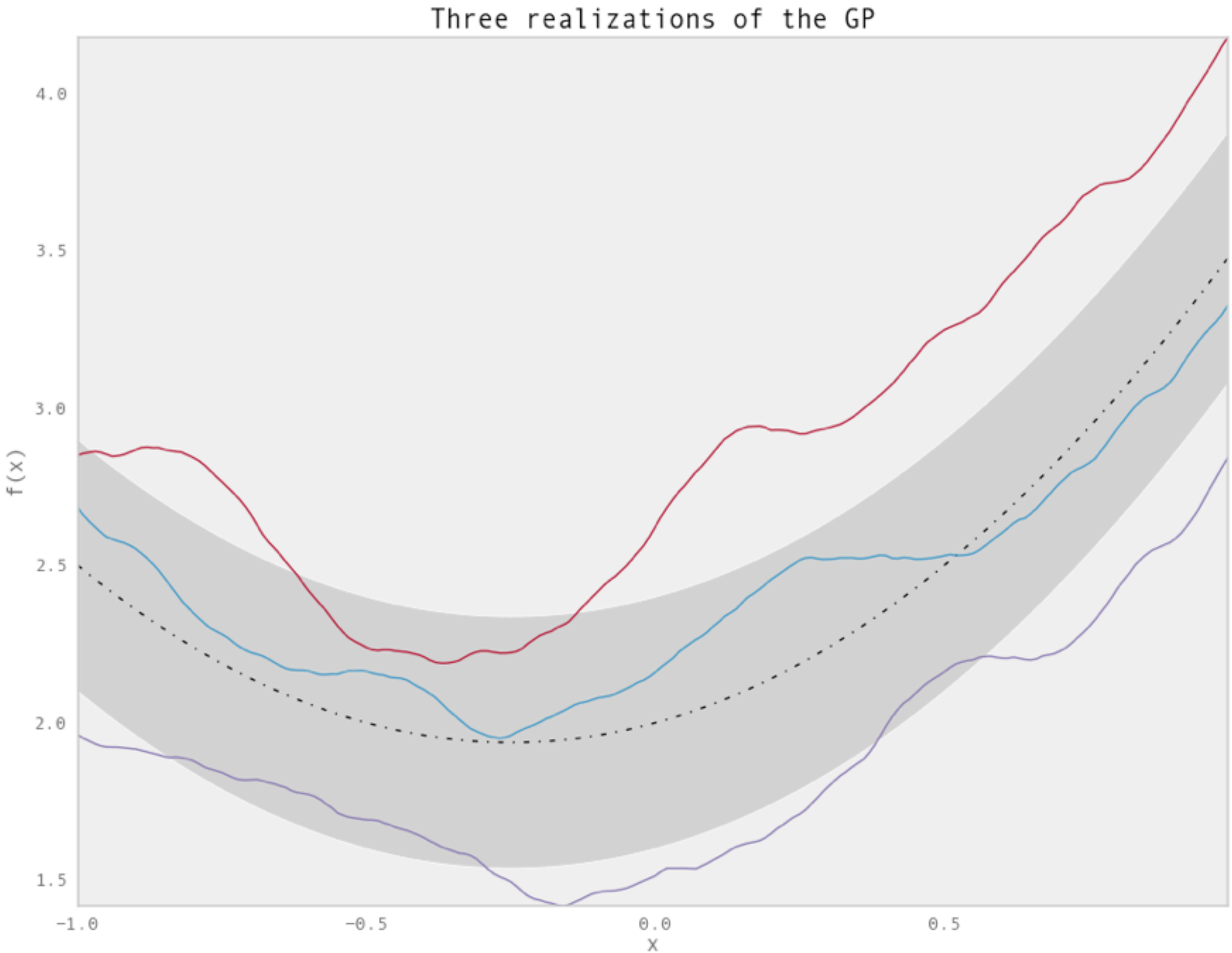

Three realizations
f(x)
x
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.0
−0.5
0.0
0.5
1.0

Gaussian process

gpexample.py

```
1  from pymc.gp import *
2
3  # Generate mean
4  def quadfun(x, a, b, c):
5      return (a * x ** 2 + b * x + c)
6
7  M = Mean(quadfun, a = 1., b = .5, c = 2.)
8
9  C = Covariance(eval_fun = matern.euclidean, diff_degree = 1.4, amp = .4, scale
   = 1.)
10
11 # Generate realizations
12 f_list=[Realization(M, C) for i in range(3)]
13
```

Line: 12 Column: 45 | Python | Soft Tabs: 4 | quadfun(x, a, b, c)


Three realizations of the GP
4.0
3.5
3.0
2.5
2.0
1.5
f(x)
−1.0
−0.5
0.0
0.5
X

observations

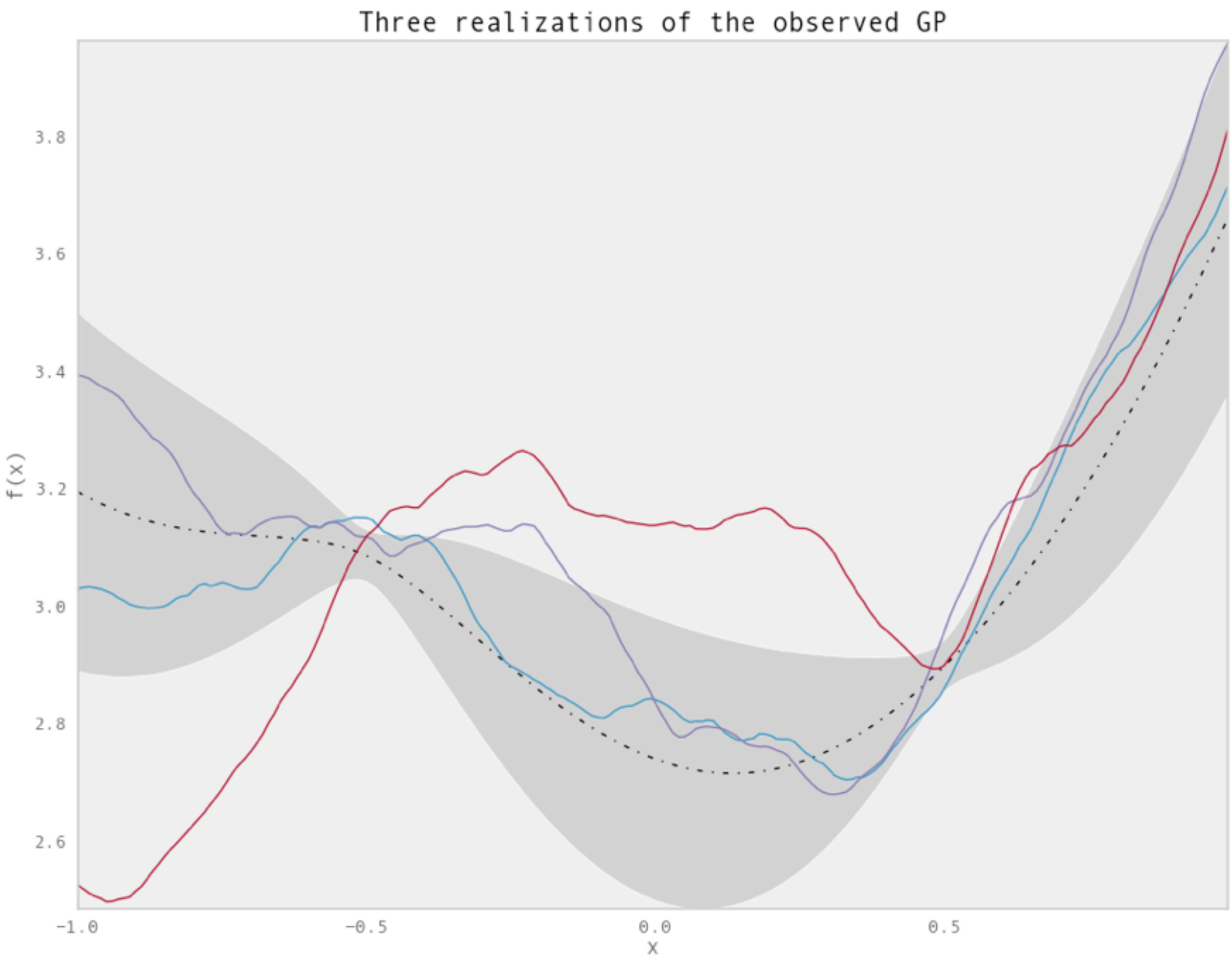
```
{x=-0.5, y=3.1},
{x=0.5, y=2.9}
```

observations

obs.py

```
1  # Import the mean and covariance
2  from mean import M
3  from cov import C
4  from pymc.gp import *
5  from numpy import *
6
7  # Impose observations on the GP
8  o = array([-.5,.5])
9  V = array([.002,.002])
10 data = array([3.1, 2.9])
11 observe(M, C, obs_mesh=o, obs_V = V, obs_vals = data)
12
13 # Generate realizations
14 f_list=[Realization(M, C) for i in range(3)]
15
```

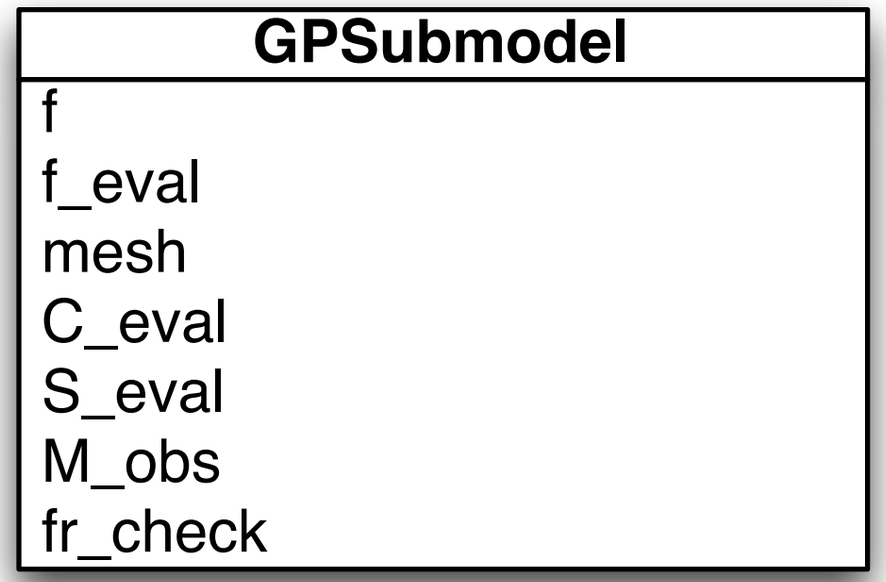
Line: 15 Column: 1 Python Soft Tabs: 4 —


Three realizations of the observed GP
3.8
3.6
3.4
3.2
3.0
2.8
2.6
f(x)
−1.0
−0.5
0.0
0.5
X

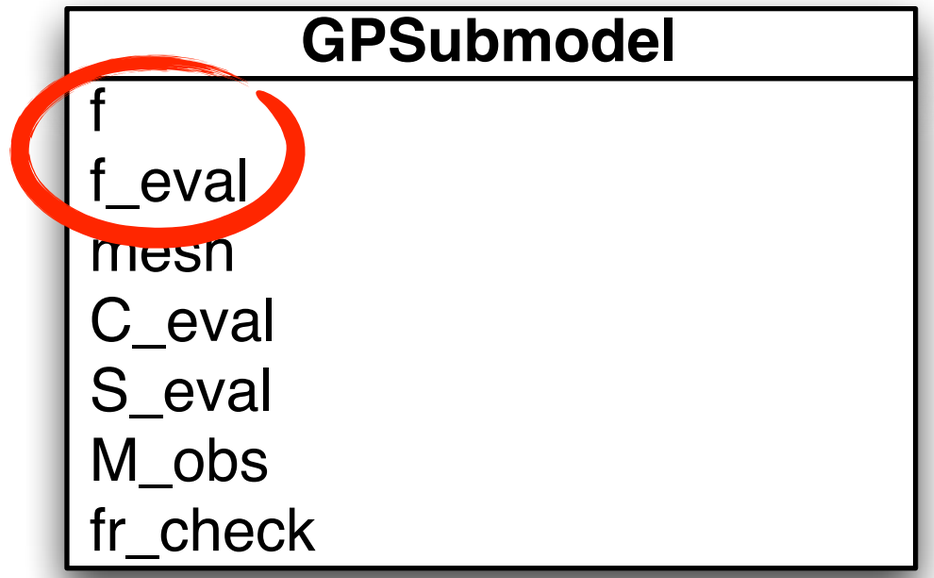
realization

$$f \sim \mathcal{GP}(m(\mathbf{x}), C(\mathbf{x}, \mathbf{x}))$$

Gaussian Process submodel



Gaussian Process submodel



```
>>> sm
<pymc.gp.gp_submodel.GPSubmodel at
0x11796a810>

>>> sm.f
<pymc.gp.gp_submodel.GaussianProcess
'sm_f' at 0x11796ab90>

>>> sm.f_eval
<pymc.gp.gp_submodel.GPEvaluation
'sm_f_eval' at 0x11796a6d0>
```

```
>>> sm.f.value
<pymc.gp.Realization.StandardRealization
at 0x10218a210>

>>> sm.f_eval.value
array([ 1.25731172,  1.19401778,
1.14649327,  1.0654201 ,  0.96870045,
        0.9051113 ,  0.82684987,
0.73403466,  0.68371817,  0.65486055,
        0.60000131,  0.52193754,
0.43451894,  0.34834653,  0.28722153,
        0.2158233 ,  0.18256193
0.66528437,  0.75319381,  0.81996597,
        0.87137425])
```

PyMCmodel.py

```
11 ¬
12     # Prior parameters of C¬
13     nu = pm.Uniform('nu', 1., 3, value=1.5)¬
14     phi = pm.Lognormal('phi', mu=.4, tau=1, value=1)¬
15     theta = pm.Lognormal('theta', mu=.5, tau=1, value=1)¬
16 ¬
17     # The covariance dtrm C is valued as a Covariance object.¬
18     @pm.deterministic¬
19     def C(eval_fun = gp.matern.euclidean, diff_degree=nu, amp=phi,
   scale=theta):¬
20         return gp.NearlyFullRankCovariance(eval_fun, diff_degree=diff_degree,
   amp=amp, scale=scale)¬
21 ¬
22     # Prior parameters of M¬
23     a = pm.Normal('a', mu=1., tau=1., value=1)¬
24     b = pm.Normal('b', mu=.5, tau=1., value=0)¬
25     c = pm.Normal('c', mu=2., tau=1., value=0)¬
26 ¬
27     # The mean M is valued as a Mean object.¬
28     def linfun(x, a, b, c):¬
29         return a * x ** 2 + b * x + c¬
30 ¬
31     @pm.deterministic¬
32     def M(eval_fun = linfun, a=a, b=b, c=c):¬
33         return gp.Mean(eval_fun, a=a, b=b, c=c)¬
34 ¬
35     # The actual observation locations¬
36     actual_obs_locs = np.linspace(-.8,.8,4)¬
37 ¬
38     if fmesh_is_obsmesh:¬
39         o = actual_obs_locs¬
40         fmesh = o¬
41     else:    ¬
42         # The unknown observation locations¬
43         o = pm.Normal('o', actual_obs_locs, 1000., value=actual_obs_locs)¬
44         fmesh = np.linspace(-1,1,n_fmesh)¬
45         ¬
```

```
    @pm.deterministic
    def M(eval_fun = linfun, a=a, b=b, c=c):
        return gp.Mean(eval_fun, a=a, b=b, c=c)

    # The actual observation locations
    actual_obs_locs = np.linspace(-.8,.8,4)

    if fmesh_is_obsmesh:
        o = actual_obs_locs
        fmesh = o
    else:
        # The unknown observation locations
        o = pm.Normal('o', actual_obs_locs, 1000., value=actual_obs_locs)
        fmesh = np.linspace(-1,1,n_fmesh)

    # The GP submodel
    sm = gp.GPSubmodel('sm',M,C,fmesh)

    # Observation variance
    V = pm.Lognormal('V', mu=-1, tau=1, value=.0001)
    observed_values = pm.rnormal(actual_obs_locs**2,10000)

    # The data d is just array-valued. It's normally distributed about
GP.f(obs_x).
    d = pm.Normal('d',mu=sm.f(o), tau=1./V, value=observed_values,
observed=True)
```

Line: 20 Column: 65 | Python | Soft Tabs: 4 | C(eval_fun = gp.matern.euclidean, diff_deg...)


GP
Step Method

$$\alpha = \frac{p(K|f')p(f'|P')q(P)}{p(K|f)p(f|P)q(P')}$$

Metropolis acceptance rate

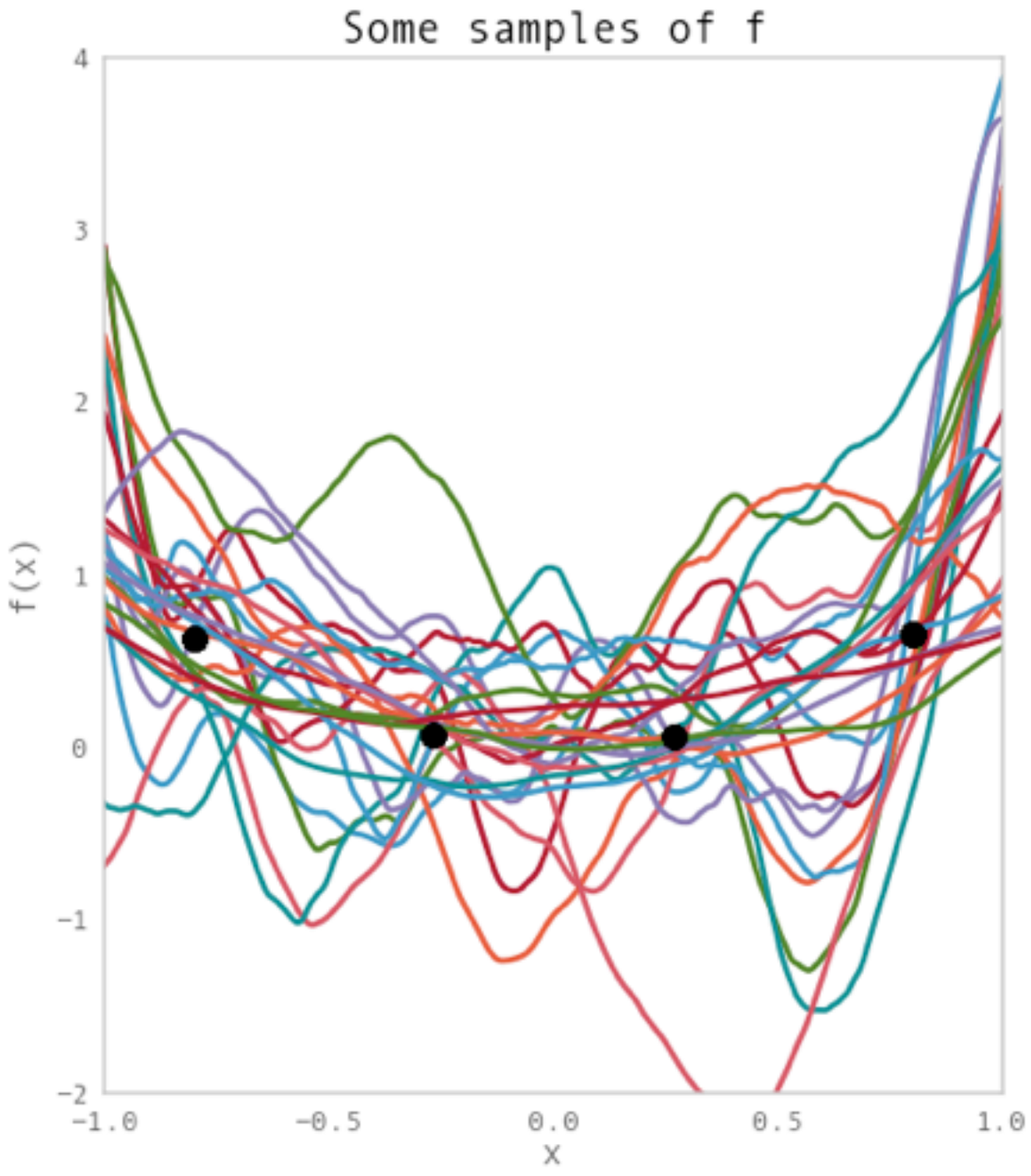
children

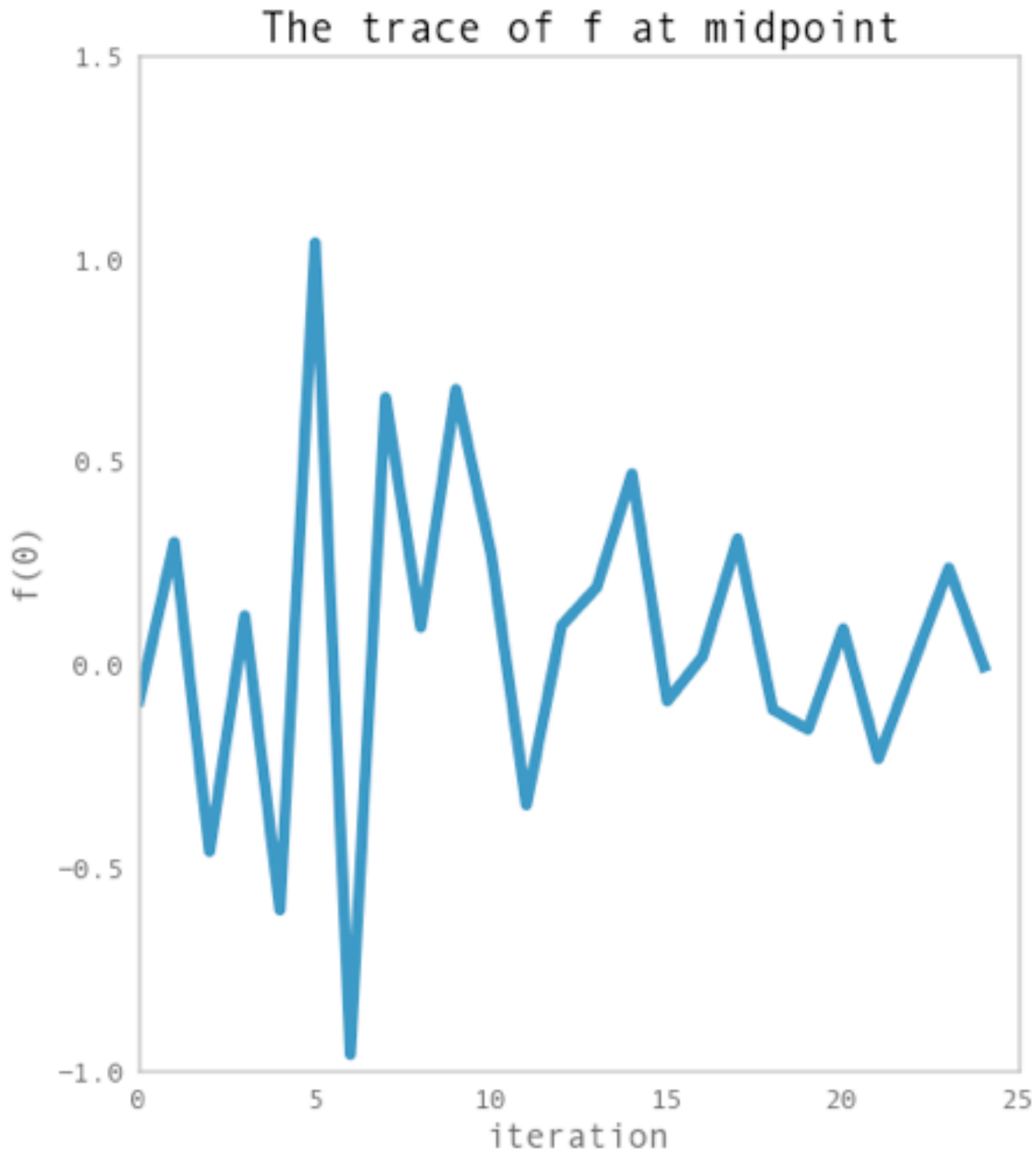
parents

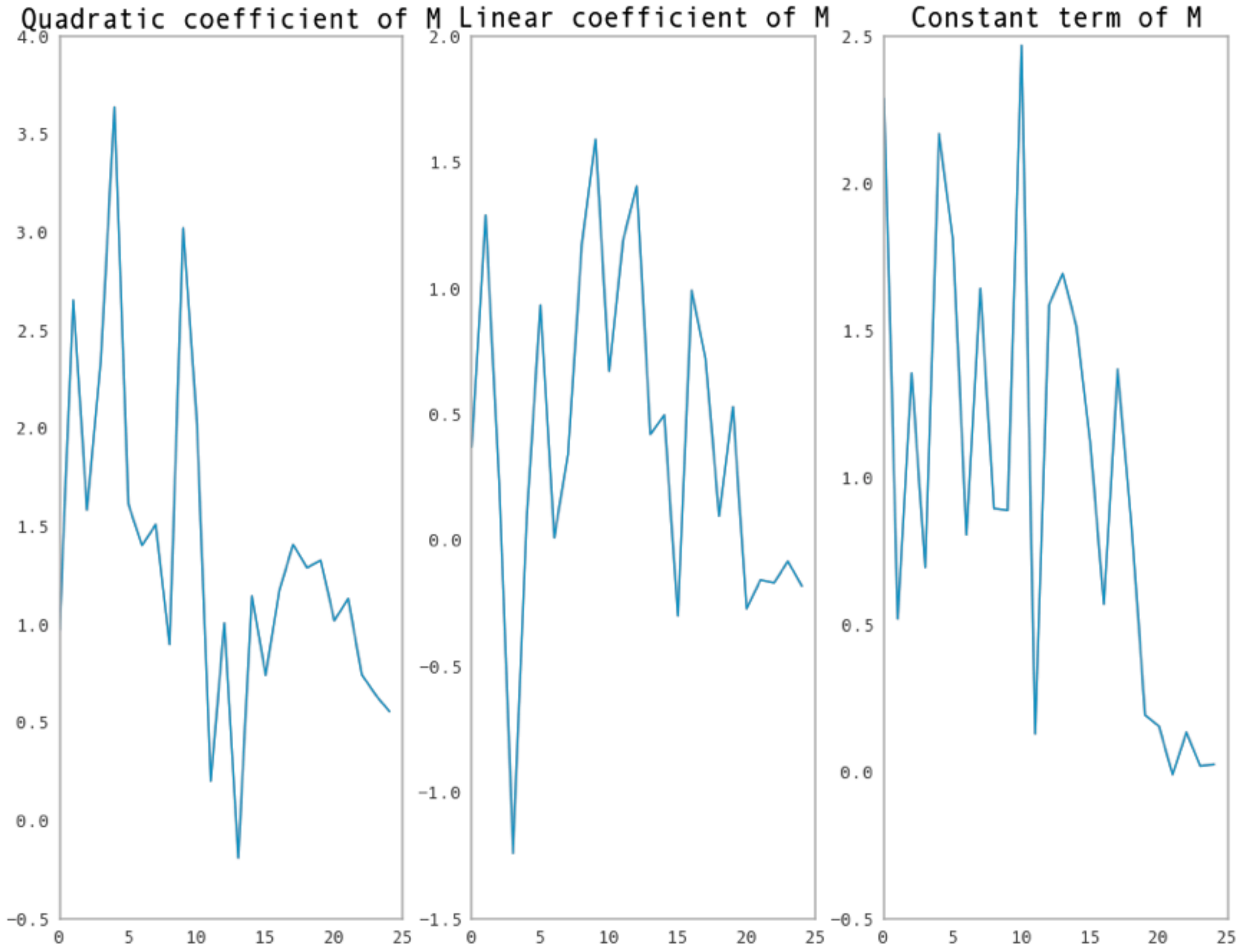
$$\alpha = \frac{p(K|f')p(f'|P')q(P)}{p(K|f)p(f|P)q(P')}$$

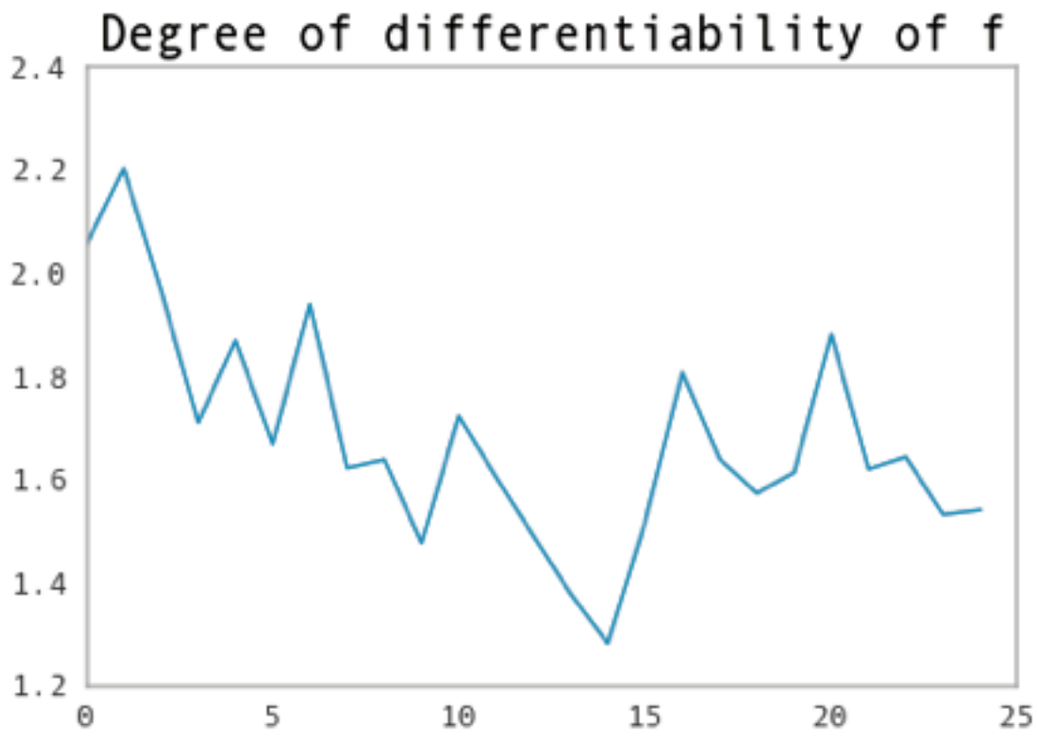
Metropolis acceptance rate

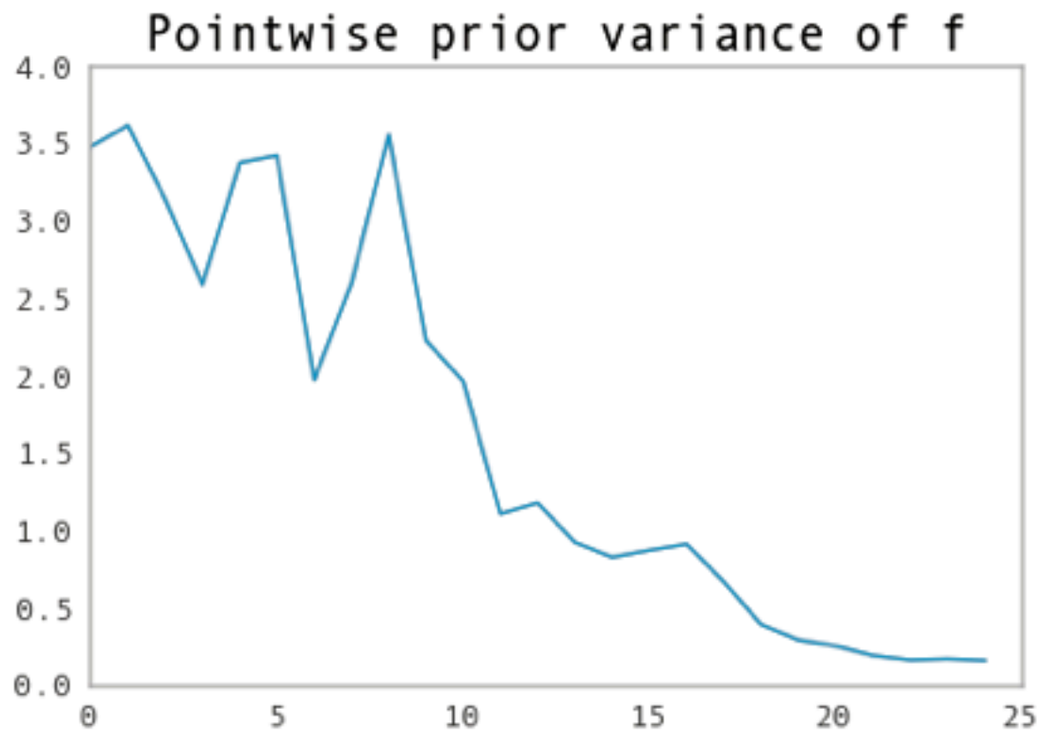
```
wrap_metropolis_for_gp_parents(some_Metropolis)
```

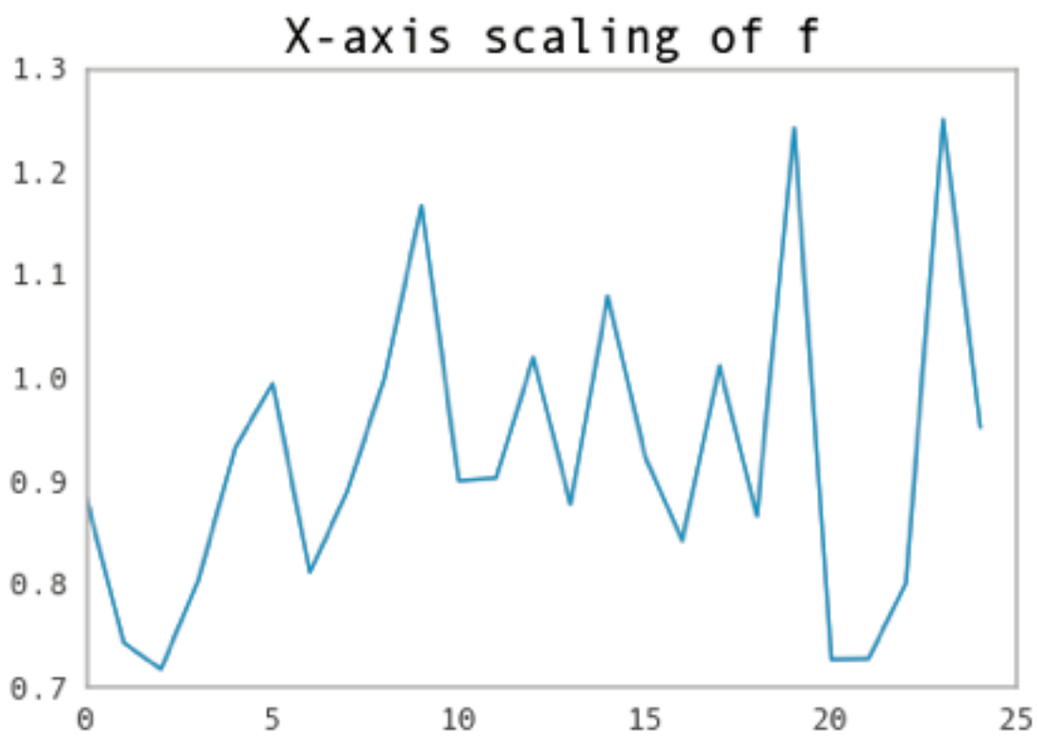

Some samples of f
f(x)
X
4
3
2
1
0
−1
−2
−1.0
−0.5
0.0
0.5
1.0

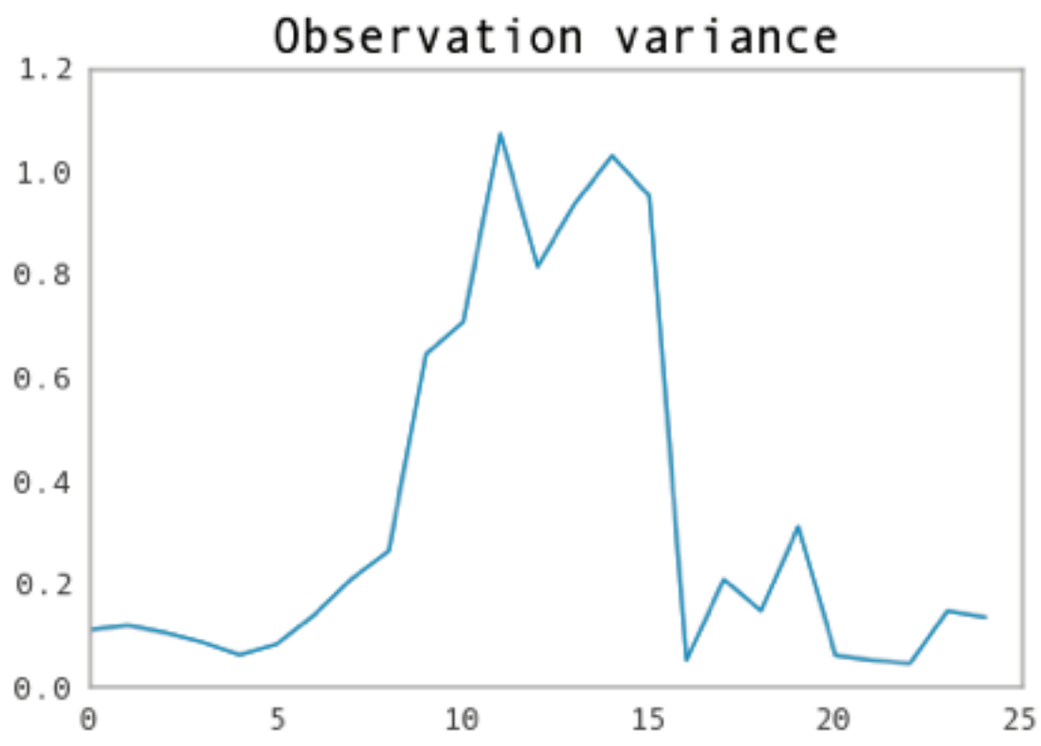

The trace of f at midpoint
f(0)
iteration
1.5
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
25


Quadratic coefficient of M
Linear coefficient of M
Constant term of M
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0
5
10
15
20
25


Degree of differentiability of f
2.4
2.2
2.0
1.8
1.6
1.4
1.2
0
5
10
15
20
25


Pointwise prior variance of f
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25


X-axis scaling of f
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0
5
10
15
20
25

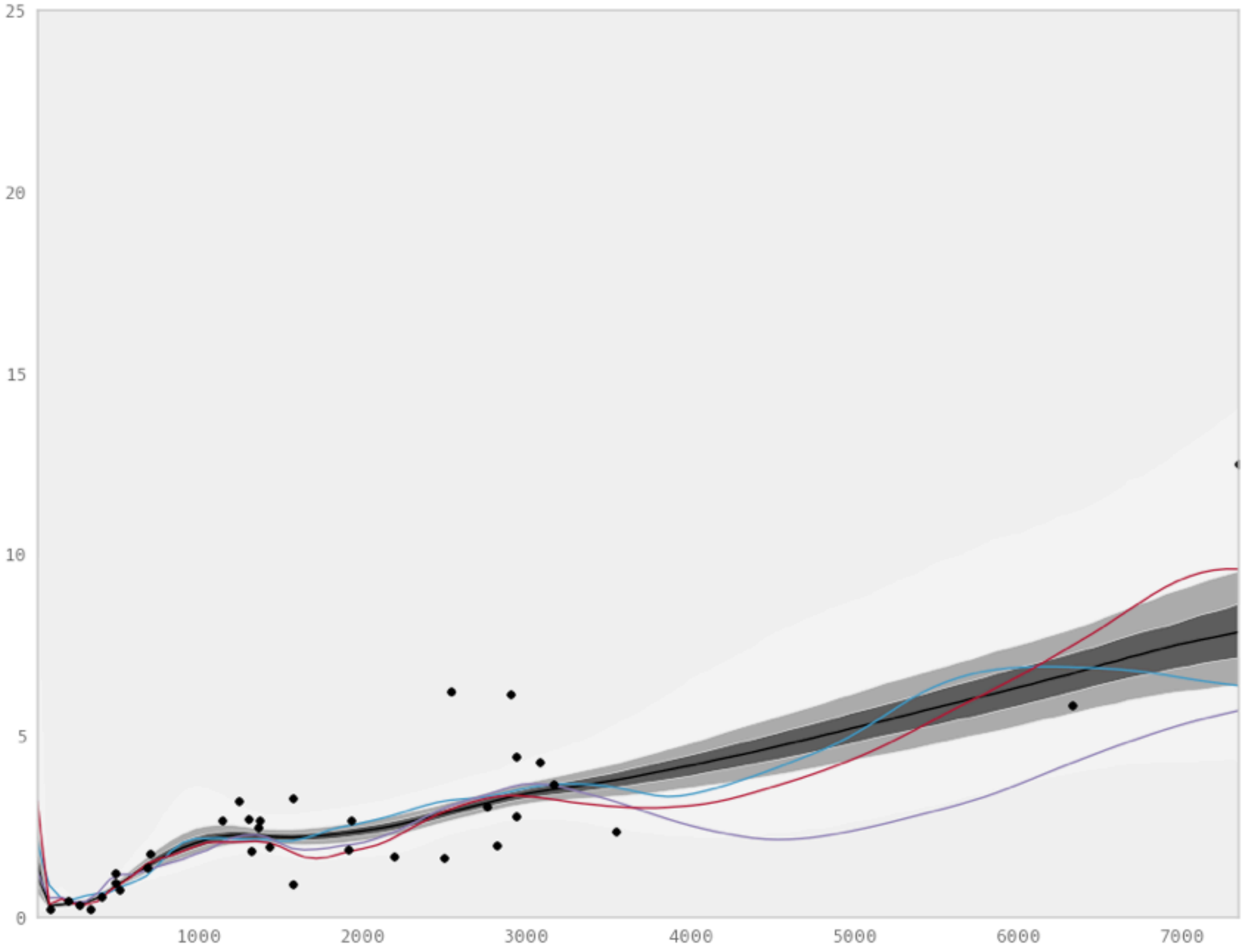

Observation variance
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25

Example

salmon recruitment

Beaverton-holt model

$$n_{t+1} = \frac{R_0 n_t}{1 + n_t / M}$$


25
20
15
10
5
0
1000
2000
3000
4000
5000
6000
7000

Gradient-based Methods

SciPy 2011

github
SOCIAL CODING

fonnesbeck | Dashboard | Inbox 0

Explore GitHub | Gist | Blog | Help

jsalvatier / gradient_samplers

Watc

Source | Commits | Network | Pull Requests (0) | Issues (0) | Graphs

Switch Branches (2) ▾ | Switch Tags (0) | Branch List

samplers for pymc that require gradients — Read more

github.com/jsalvatier/gradient_samplers

HTTP | Git Read-Only | https://github.com/jsalvatier/gradient_samplers | Read-Only access

added error checking to hessian estimation

jsalvatier (author)
March 27, 2011

commit
tree
parent

gradient_samplers /

name	age	message
gradient_samplers/	March 27, 2011	added error checking to hessian estimation [jsalvatier
README	February 13, 2011	first commit [jsalvatier]
setup.py	March 03, 2011	added better hessian approximation [jsalvatier]

In the sandbox

multiprocessing
multi-model inference
RJMCMC
Dirichlet processes
EM