PyMC's Big Adventure

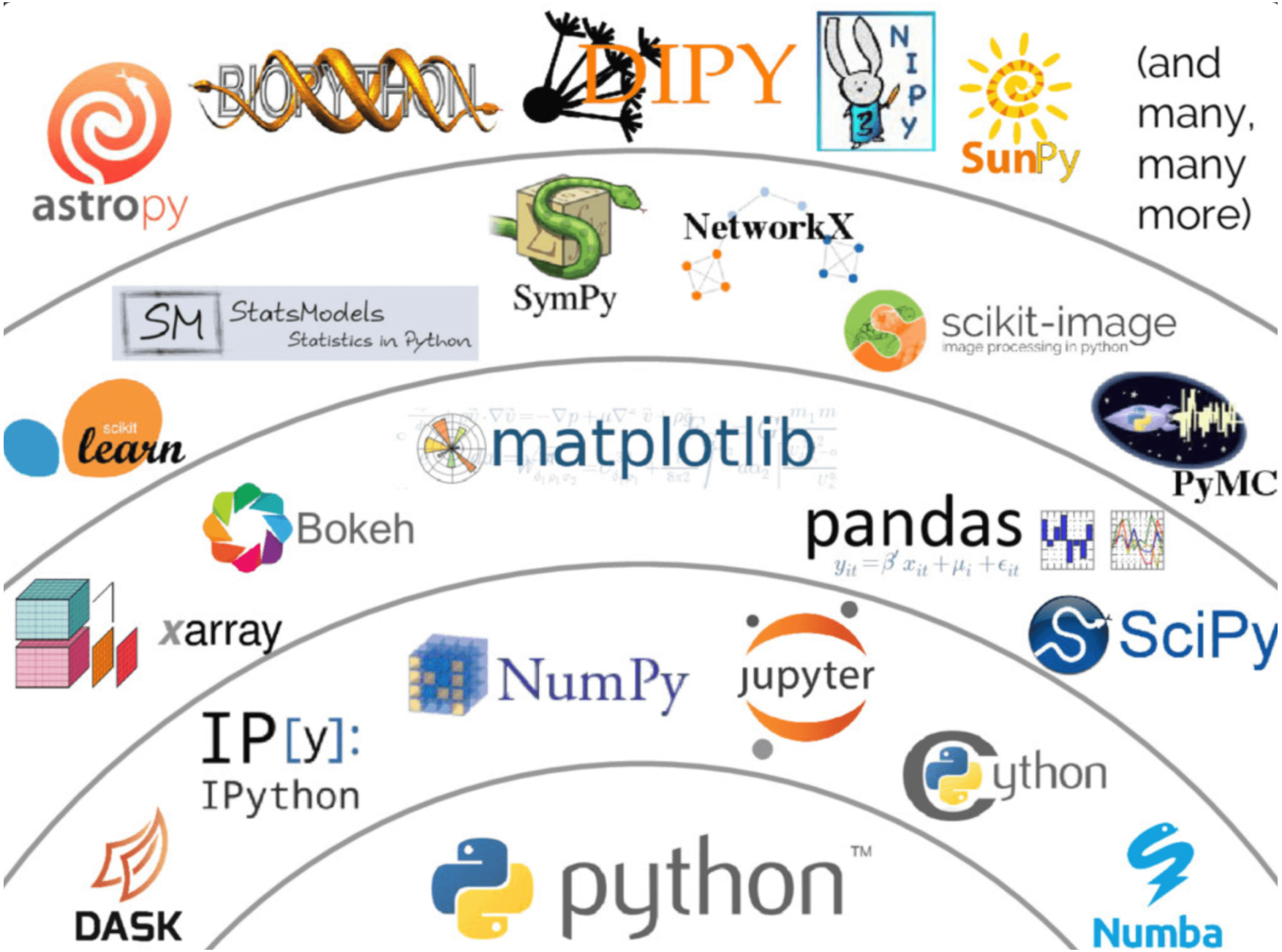
Lessons Learned from the Development of Open-source Software for Probabilistic Programming

Chris Fonnesbeck, Vanderbilt University Medical Center

Machine Learning Open Source Software 2018: Sustainable communities

- started in 2003
- **probabilistic programming** framework
- based on Theano
- implements *next generation* Bayesian inference methods
- >100 contributors

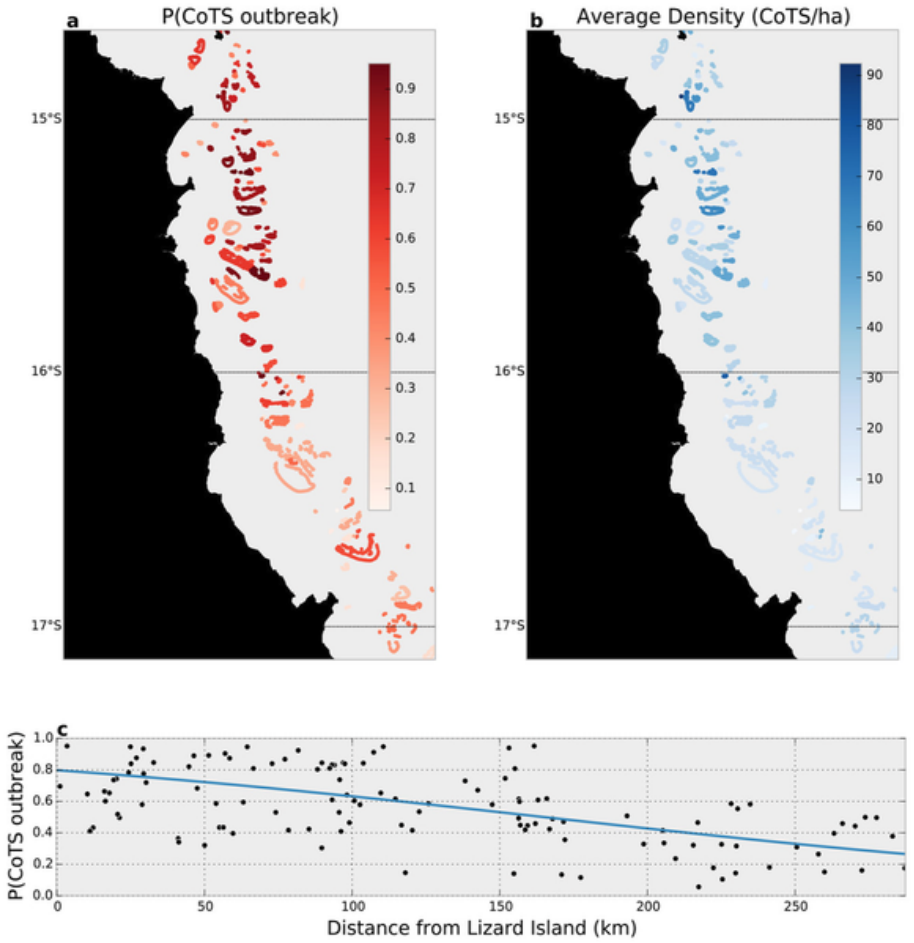
Salvatier, Wiecki and Fonnesbeck (2016)


astropy
BIOPYTHON
DIPY
NIPY
SunPy
(and many, many more)
SymPy
NetworkX
SM StatsModels Statistics in Python
scikit-image image processing in python
scikit learn
matplotlib
PyMC
Bokeh
pandas
xarray
NumPy
jupyter
SciPy
IP[y]: IPython
Cython
DASK
python™
Numba

PyMC3 in the Wild

- Astronomy
- Biostatistics
- Ecology
- Chemistry
- Finance
- Economics
- Sabermetrics
- Marketing
- Actuarial Science

Joint estimation of crown of thorns *Acanthaster planci* densities on the Great Barrier Reef

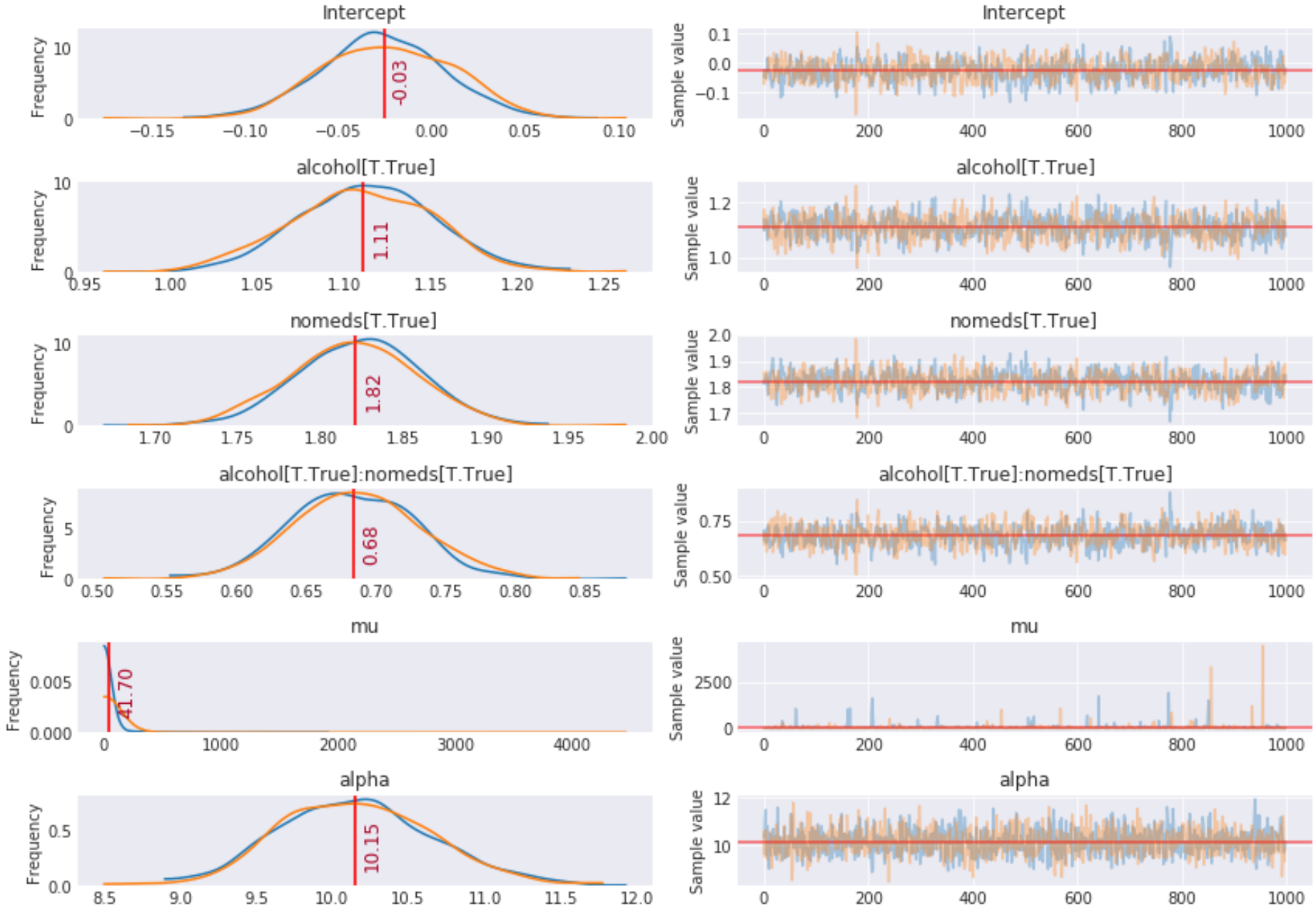


MacNeil *et al.* 2016

In [19]:
```
import pymc3 as pm

with pm.Model() as framing_model:

    # Baseline probability
    μ = pm.Normal('μ', 2, sd=5)
    σ = pm.Exponential('σ', 1)
    z = pm.Normal('z', 0, 1, shape=n_catchers)
    # Non-centered parameterization
    θ = pm.Deterministic('θ', μ + z*σ)

    # Called strike rate as function of distance to strike zone
    p = pm.math.invlogit(θ[catcher_ind])

    called_strike = pm.Bernoulli('called_strike', p, observed=strike)
```

In [20]:
```
with framing_model:
    samples = pm.sample(1000, tune=1000, njobs=2)
```

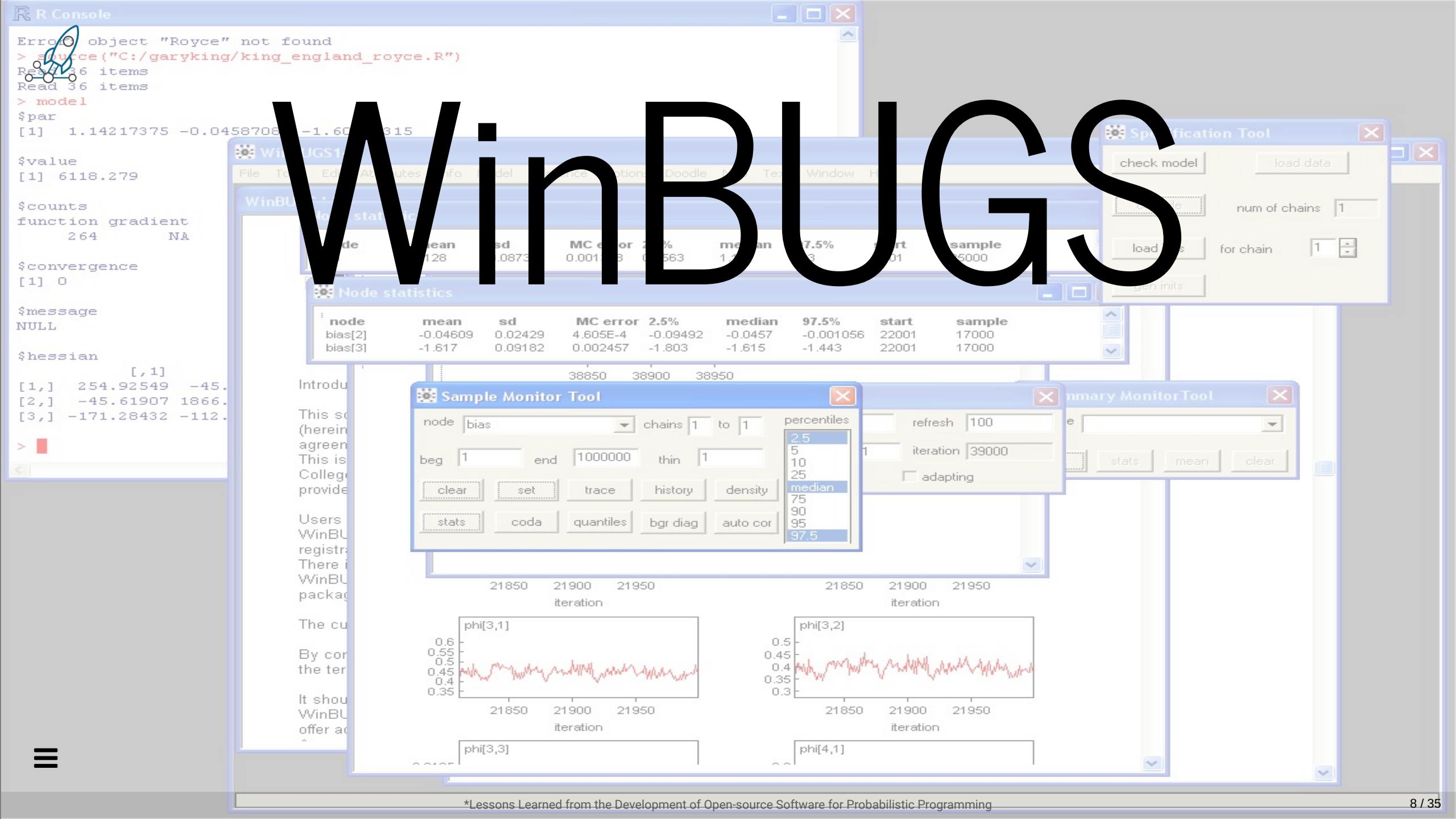
```
INFO:pymc3:Auto-assigning NUTS sampler...
INFO:pymc3:Initializing NUTS using jitter+adapt_diag...
INFO:pymc3:Multiprocess sampling (2 chains in 2 jobs)
INFO:pymc3:NUTS: [z, σ, μ]
Sampling 2 chains: 100%|██████████| 4000/4000 [01:49<00:00, 58.81draws/s]
```


Intercept
Frequency
-0.03
Sample value
alcohol[T.True]
1.11
nomeds[T.True]
1.82
alcohol[T.True]:nomeds[T.True]
0.68
mu
41.70
alpha
10.15

Why PyMC?

Bayesian modeling for applied users ☺

WinBUGS


R Console
> source("C:/garyking/king_england_royce.R")
Read 36 items
Read 36 items
> model
$par
$value
[1] 6118.279
$counts
function gradient
264 NA
$convergence
[1] 0
$message
NULL
$hessian
Node statistics
node mean sd MC error 2.5% median 97.5% start sample
bias[2] -0.04609 0.02429 4.605E-4 -0.09492 -0.0457 -0.001056 22001 17000
bias[3] -1.617 0.09182 0.002457 -1.803 -1.615 -1.443 22001 17000
Sample Monitor Tool
node bias chains 1 to 1
beg 1 end 1000000 thin 1
clear set trace history density
stats coda quantiles bgr diag auto cor
percentiles
2.5
5
10
25
median
75
90
95
97.5
check model
load data
num of chains 1
for chain 1
gen inits
refresh 100
iteration 39000
adapting
stats mean clear
iteration
phi[3,1]
phi[3,2]
phi[3,3]
phi[4,1]

```
model {
    for (j in 1:J){
      y[j] ~ dnorm (theta[j], tau.y[j])
      theta[j] ~ dnorm (mu.theta, tau.theta)
      tau.y[j] <- pow(sigma.y[j], -2)
    }
    mu.theta ~ dnorm (0.0, 1.0E-6)
    tau.theta <- pow(sigma.theta, -2)
    sigma.theta ~ dunif (0, 1000)
  }
```

PyMC 1.0

MCMC.py

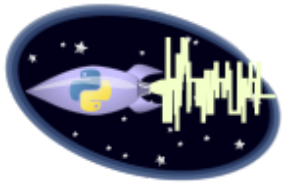
```
229  class Parameter(Node):
230      """
231      Parameter class extends Node class, and represents a variable to be
232      estimated using MCMC sampling. Generates candidate values using a
233      random walk algorithm. The default proposal is a standard normal
234      density, though any zero-centered distribution may be substituted. The
235      proposal is usually adapted by the sampler to achieve an optimal
236      acceptance rate (between 20 and 50 percent).
237      """
238
239      def __init__(self,name,              # Parameter name
240                   init_val,               # Initial value
241                   sampler,                # Sampler
242                   dist='normal',          # Proposal distribution
243                   scale=1,                # Proposal scale hyperparameter
244                   minval=-inf,maxval=inf,     # Min/max values
245                   maxsum=inf):            # Max value of sum of parameters
246          'Class initialization'
247
248          'Initialize superclass'
249          Node.__init__(self,name,sampler)
250
251          'Parameter bounds'
252          self._min = minval
253          self._max = maxval
254          self._maxsum = maxsum
255
256          'Counter for rejected proposals; used for adaptation.'
257          self._rejected = 0
```

And Then There Were Three ...

Anand Patil

Me

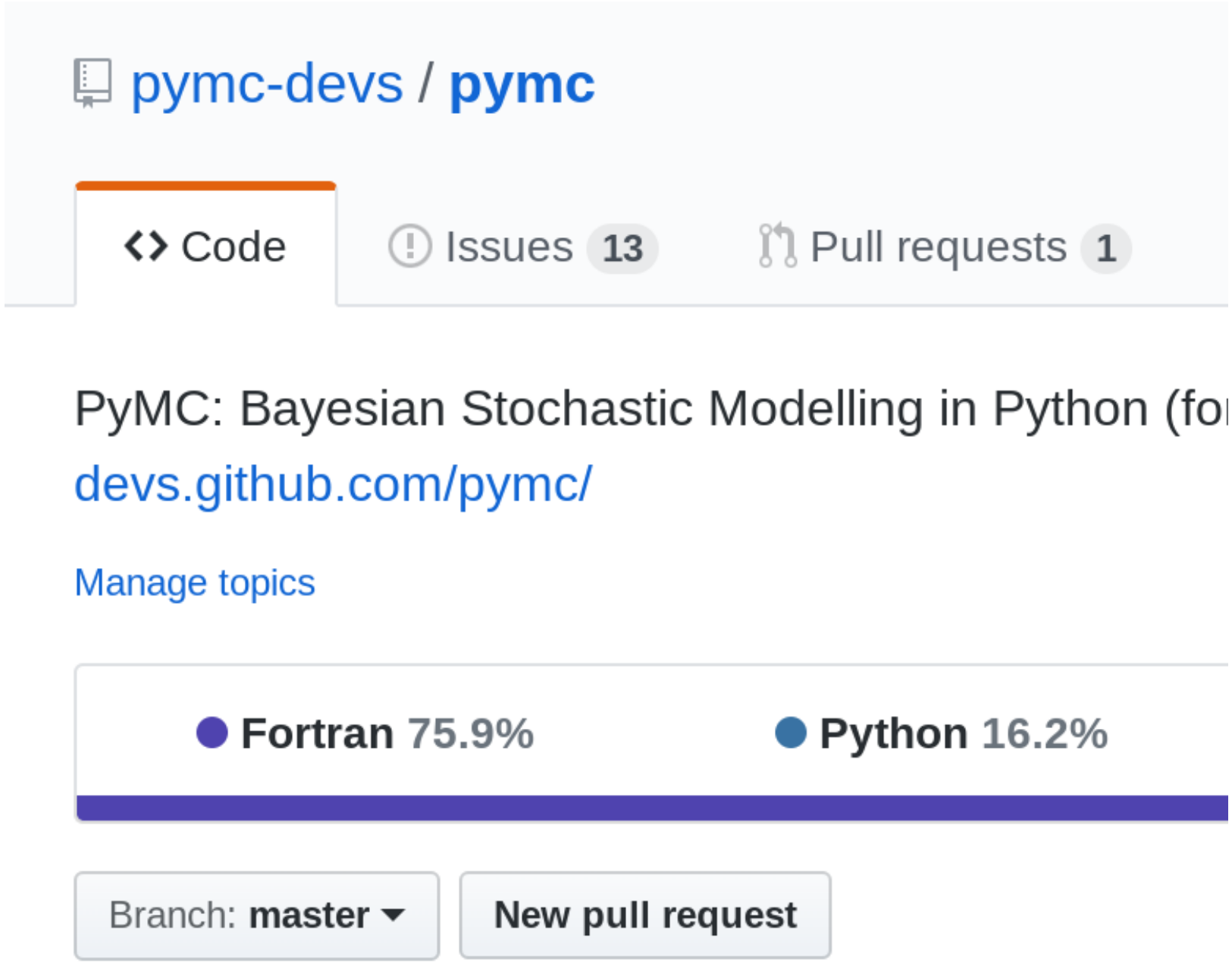
David Huard



PyMC2

- Suite of well-documented statistical distributions.
- NumPy-based
- Gaussian processes module
- Robust sampling loops
- Suite of convergence diagnostics
- Extensible: custom step methods and unusual probability distributions
- f2py FORTRAN extensions

Patil, A., D. Huard, and C.J. Fonnesbeck. 2010. Journal of Statistical Software 35 [4].


pymc-devs / pymc
Code
Issues 13
Pull requests 1
PyMC: Bayesian Stochastic Modelling in Python (fo
devs.github.com/pymc/
Manage topics
Fortran 75.9%
Python 16.2%
Branch: master
New pull request

Hamiltonian Monte Carlo

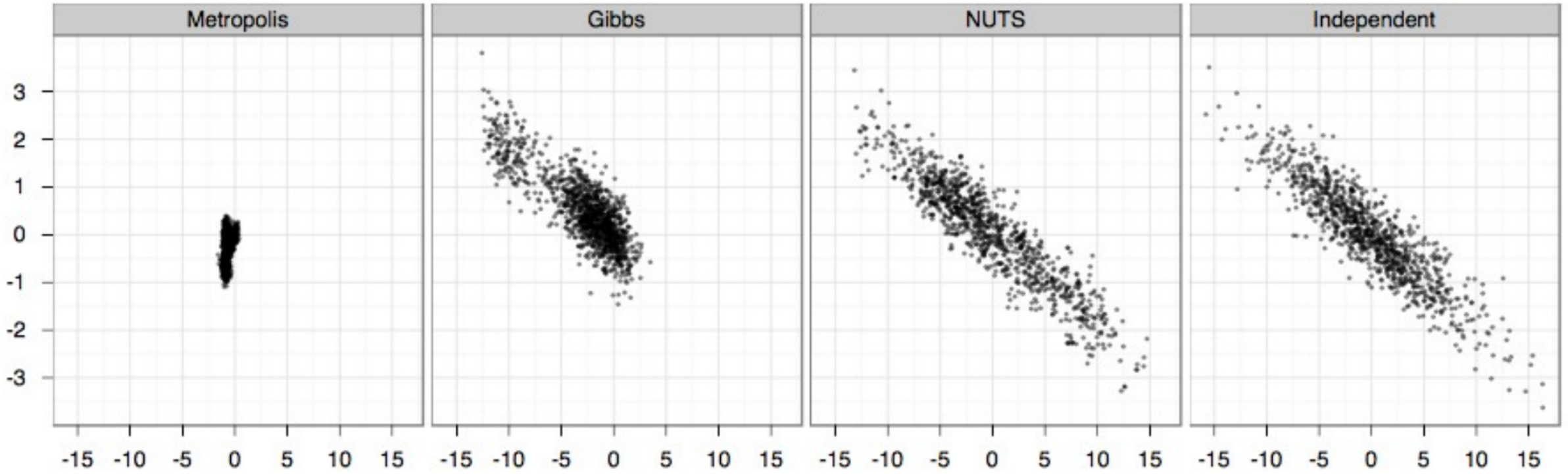
Uses a *physical analogy* of a frictionless particle moving on a hyper-surface

Requires an *auxiliary variable* to be specified

- position (unknown variable value)
- momentum (auxiliary)

$$\mathcal{H}(s, \phi) = E(s) + K(\phi) = E(s) + \frac{1}{2}(\sum_i)\phi_i^2$$

Efficient MCMC sampling



Hoffman and Gelman, 2014

"I’m not sure why this approach seems neglected. It might be that research incentives don’t reward such generally applicable research, or that MCMC researchers do not see how simplified MCMC could dramatically improve the productivity of statistics, or perhaps researchers haven’t realized how automatic differentiation can democratize these algorithms."

John Salvatier

Calculating Gradients in Theano

```
>>> from theano import function, tensor as tt
>>> x = tt.dmatrix('x')
>>> s = tt.sum(1 / (1 + tt.exp(-x)))
>>> gs = tt.grad(s, x)
>>> dlogistic = function([x], gs)
>>> dlogistic([[3, -1],[0, 2]])
array([[ 0.04517666,  0.19661193],
       [ 0.25      ,  0.10499359]])
```

PyMC Devs, The Next Generation

John Salvatier

Me

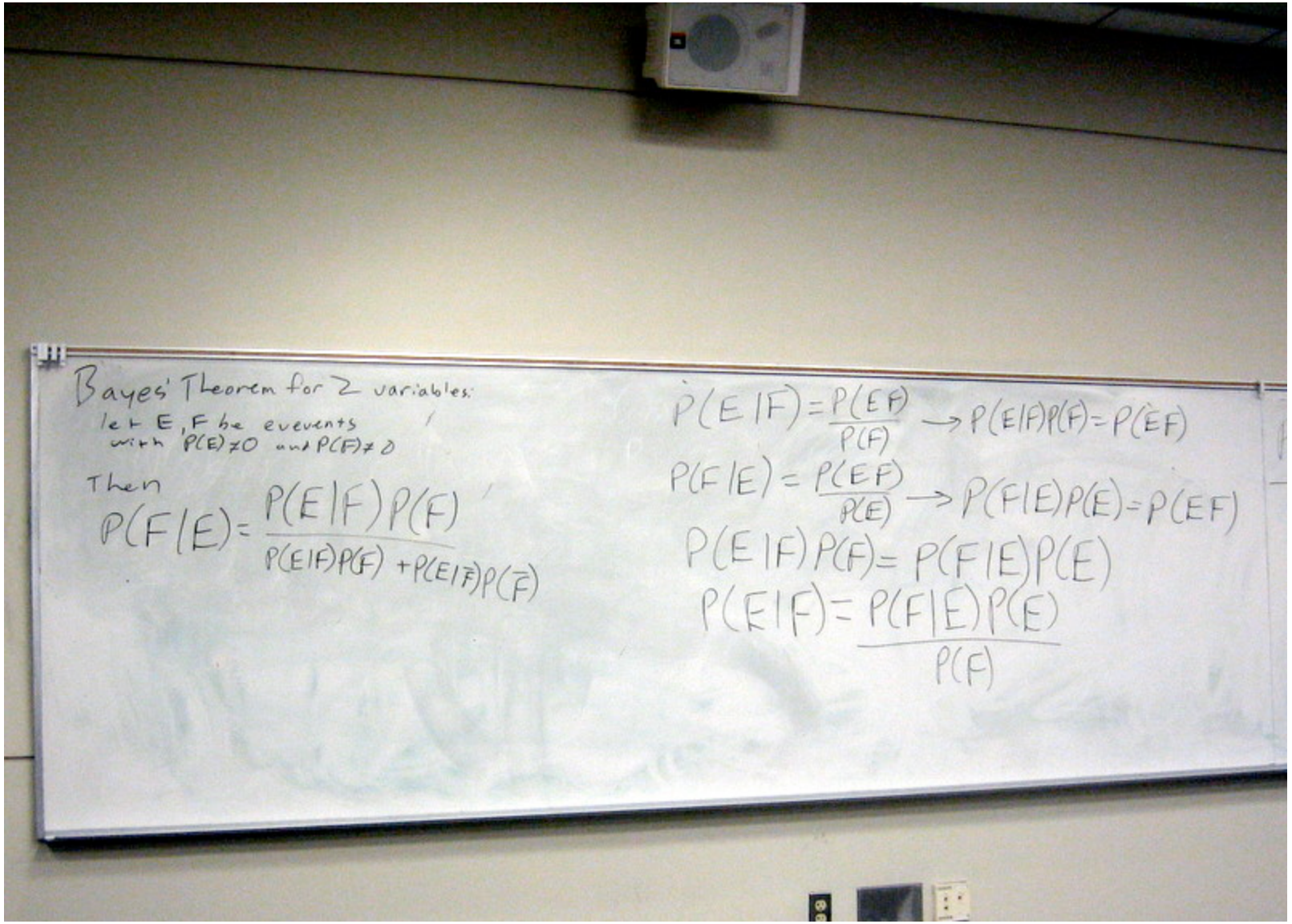
Thomas Wiecki

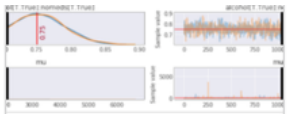
Regression Model

$$\beta_{0,i} \sim N(0, 10000)$$
$$\beta_1 \sim N(0, 10000)$$
$$\sigma \sim C^+(5)$$
$$\theta_i = \beta_{0,i} + \beta_1 x_i$$
$$y_i \sim N(\theta_i, \sigma)$$

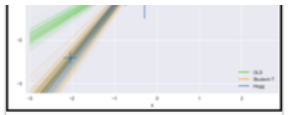
Regression Model

```
with Model() as unpooled_model:

    β0 = Normal('β0', 0, sd=1e5, shape=counties)
    β1 = Normal('β1', 0, sd=1e5)
    σ = HalfCauchy('σ', 5)

    θ = β0[county] + β1*floor

    y = Normal('y', θ, sd=σ, observed=log_radon)
```

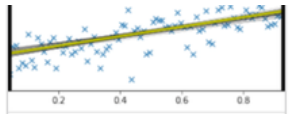

Bayes' Theorem for 2 variables:
let E, F be eveents
with P(E)≠0 and P(F)≠0
Then
P(F|E) = P(E|F)P(F) / P(E|F)P(F) + P(E|F̄)P(F̄)
P(E|F) = P(EF)/P(F) → P(E|F)P(F) = P(EF)
P(F|E) = P(EF)/P(E) → P(F|E)P(E) = P(EF)
P(E|F)P(F) = P(F|E)P(E)
P(E|F) = P(F|E)P(E) / P(F)



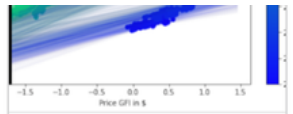
GLM: Poisson Regression



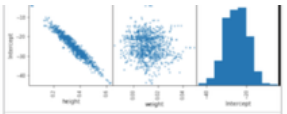
GLM: Robust Regression with Outlier Detection



GLM: Robust Linear Regression



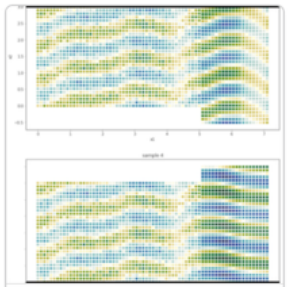
Rolling Regression



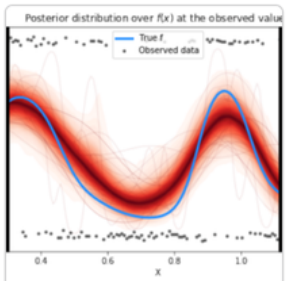
GLM: Linear Regression

Hierarchical Partial Pooling

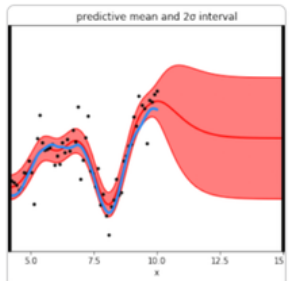
Gaussian Processes



Kronecker Structured Covariances

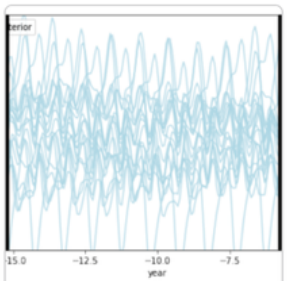


Latent Variable Implementation

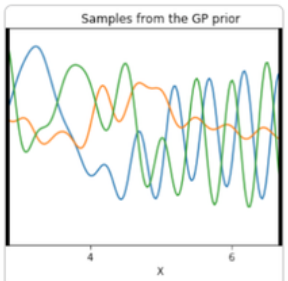


Marginal Likelihood Implementation

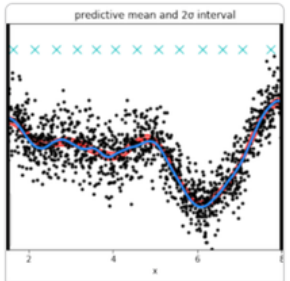
Example: CO2 at Mauna Loa



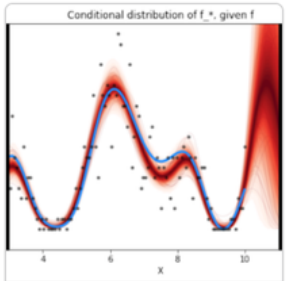
Example: Mauna Loa CO$_2$ continued



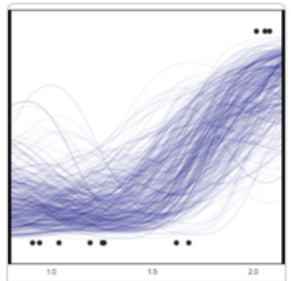
Mean and Covariance Functions



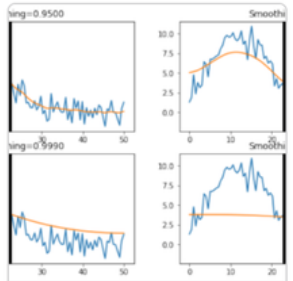
Sparse Approximations



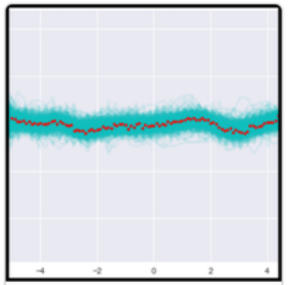
Student-t Process



Gaussian Process Regression and Classification



Gaussian Process (GP) smoothing



Gaussian Processes

MCMC Sampling

```
with pitch_framing_model:
    trace = pm.sample(1000, tune=1000)

Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [σ_log__, ψ, μ]
100%|          | 2000/2000 [00:42<00:00, 47.42it/s]
```

Variational Inference

```
with neural_network:
    approx = pm.fit(n=30000, method='fullrank_advi')

Average Loss = 1,207.6: 100%|          | 30000/30000 [27:41
Finished [100%]: Average Loss = 1,206.2
```

MILA and the future of Theano

Pascal Lamblin

Dear users and developers,

After almost ten years of development, we have the regret to announce that we will put an end to our Theano development after the 1.0 release, which is due in the next few weeks. We will continue minimal maintenance to keep it working for one year, but we will stop actively implementing new features. Theano will continue to be available afterwards, as per our engagement towards open source software, but MILA does not commit to spend time on maintenance or support after that time frame.

The software ecosystem supporting deep learning research has been evolving quickly, and has now reached a healthy state: open-source software is the norm; a variety of frameworks are available, satisfying needs spanning from exploring novel ideas to deploying them into production; and strong industrial players are backing different software stacks in a stimulating competition.

We are proud that most of the innovations Theano introduced across the years have now been adopted and perfected by other frameworks. Being able to express models as mathematical expressions, rewriting

Chris Fonnesbeck
@fonnesbeck

The #PyMC3 devs are frequently asked about what's going to happen to the project now that Theano is winding down. Here is a summary of where things are headed.

Theano, TensorFlow and the Future of PyMC – PyMC Devel…

PyMC3 is an open-source library for Bayesian statistical modeling and inference in Python, implementing gradient-based Markov chain Monte…

medium.com

2:50 PM - 17 May 2018


mxnet


Chainer

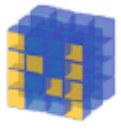

PYTORCH


CANNOT
PREDICT
NOW

NUMFOCUS

OPEN CODE = BETTER SCIENCE

SPONSORED PROJECTS

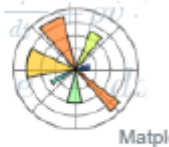

NumPy


jupyter


Open Journals


SymPy


julia


Matplotlib


IPython


rOpenSci

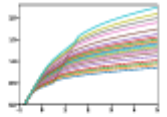

FENICS PROJECT


PyTables


pandas


nteract


QuantEcon


Econ-ARK


AstroPy


Stan


将軍


software carpentry

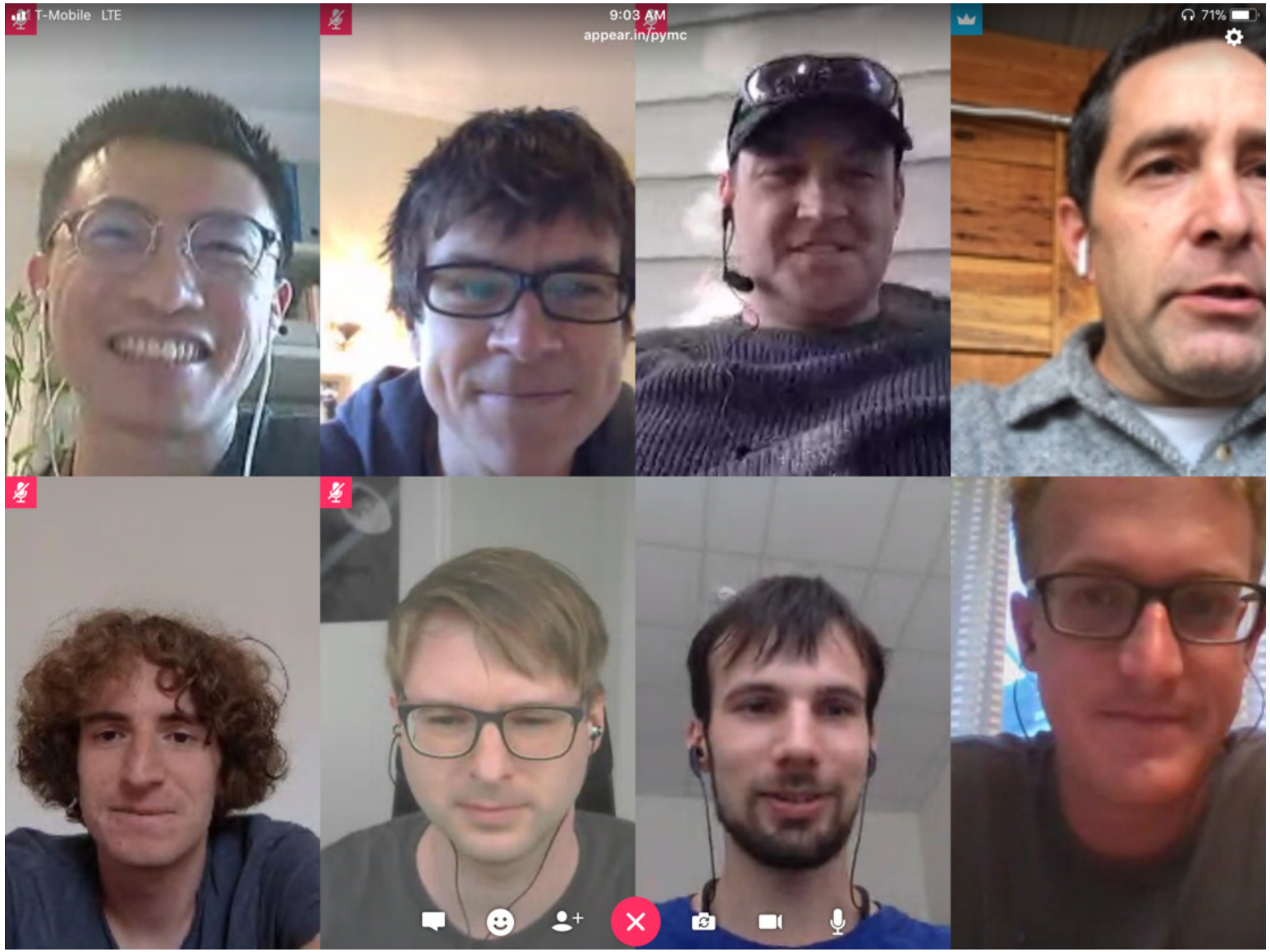

yt


sunpy


PyMC3


DATA CARPENTRY


Google
SUMMER
of
CODE


T-Mobile LTE
9:03 AM
appear.in/pymc
71%

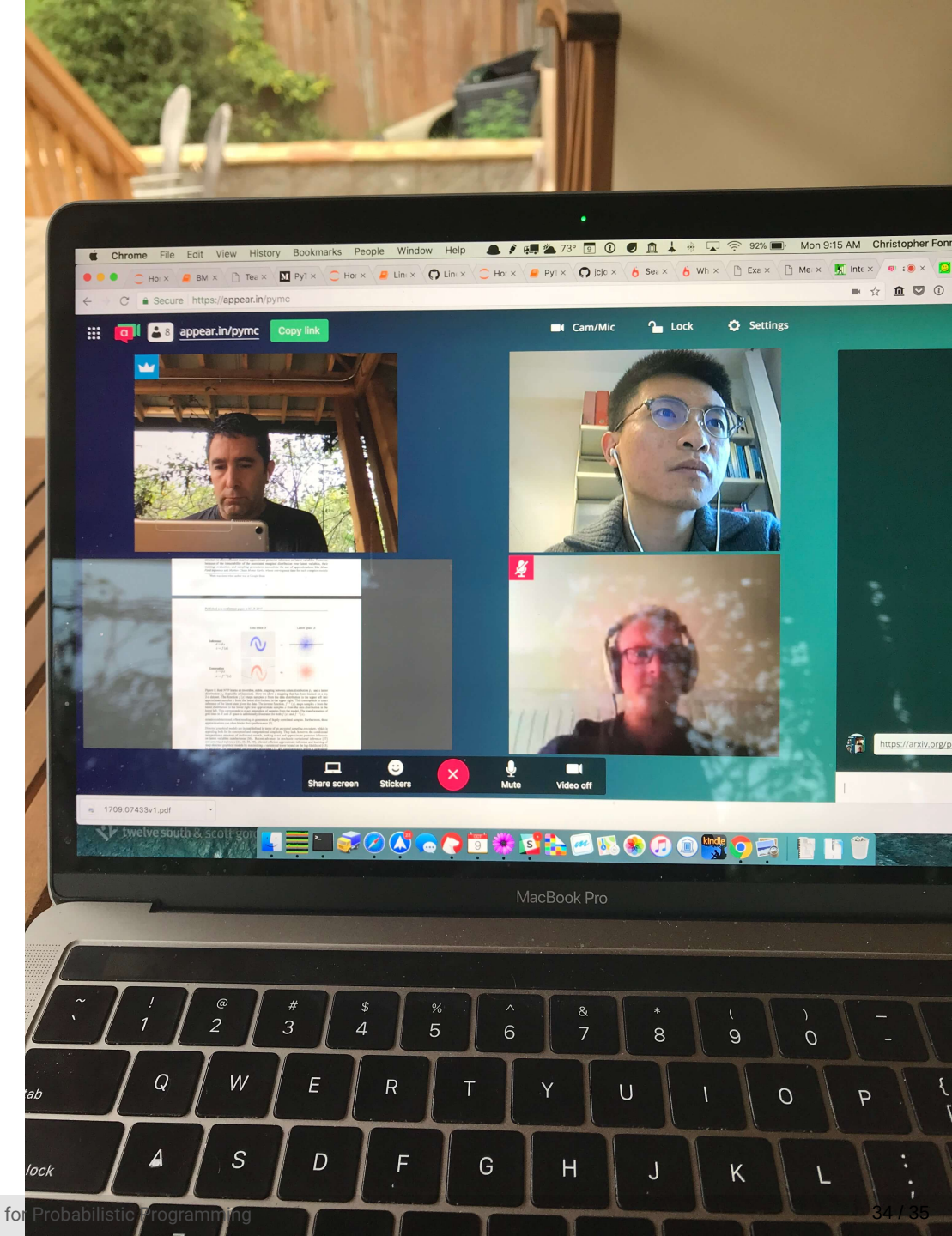
PyData NYC 2017

In [19]:
```
summary(trace)
```

```
alpha:

  Mean             SD               MC Error         95% HPD interval
  -------------------------------------------------------------------

  3.002            0.220            0.003            [2.582, 3.440]

  Posterior quantiles:
  2.5              25               50               75               97.5
  |--------------|================|================|--------------|

  2.579            2.851            3.000            3.150            3.440


beta:

  Mean             SD               MC Error         95% HPD interval
  -------------------------------------------------------------------

  3.968            0.238            0.003            [3.501, 4.423]

  Posterior quantiles:
  2.5              25               50               75               97.5
  |--------------|================|================|--------------|

  3.502            3.809            3.966            4.129            4.424


s:

  Mean             SD               MC Error         95% HPD interval
  -------------------------------------------------------------------

  2.021            0.149            0.002            [1.734, 2.306]

  Posterior quantiles:
  2.5              25               50               75               97.5
  |--------------|================|================|--------------|

  1.743            1.918            2.014            2.117            2.322
```

Interaction and Communication

- Monthly lab meetings
- Journal club
- Face-to-face meetings
- Slack channels
- Discourse site




Enjoy
Coca-Cola
Trade-mark ®
Bayes CARRY OUT