BEYOND WINBUGS
SURVEYING THE FUTURE OF BAYESIAN STATISTICAL MODELING
CHRIS FONNESBECK, VANDERBILT BIOSTATISTICS
WORKSHOP: ADVANCING SOFTWARE FOR ECOLOGICAL FORECASTING
CHAMPAIGN, IL
MARCH 25-27, 2014

Organizers

» David LeBauer (Univ. Illinois, Urbana-Champaign)

» Perry de Valpine (Univ. California, Berkeley)

» Matthew Smith (Microsoft Research)
"INFERENCES ENGINES"

» Perry de Valpine UC at Berkeley NIMBLE
» Matthew Smith Microsoft Research Filzbach
» Mike Dietze Boston University PEcAn
» Andrew Thomas Rolf Nevanlinna Institute OpenBUGS
» Ed Ionides University of Michigan POMP
» Chris Fonnesbeck Vanderbilt University PyMC
"INFERENC E ENGINES" (CONTINUED)

» Martyn Plummer WHO International Agency for Research on Cancer JAGS

» Andy Fox NEON DART

» Janine Illian University of St Andrews R-INLA

» Anders Nielsen National Institute of Aquatic Resources ADMB

» Daniel Lee Columbia University STAN
$$\Pr(\theta|y) = \frac{\Pr(y|\theta)\Pr(\theta)}{\Pr(y)}$$

- Posterior Probability
- Likelihood of Observations
- Prior Probability
- Normalizing Constant
"... the Bayesian approach is attractive because it is useful. Its usefulness derives in large measure from its simplicity. Its simplicity allows the investigation of far more complex models than can be handled by the tools in the classical toolbox."

Link and Barker 2010
MARKOV CHAIN MONTE CARLO
MCMC

Markov chain Monte Carlo simulates a Markov chain for which some function of interest is the unique, invariant, limiting distribution.
This is guaranteed when the Markov chain is constructed that satisfies the **detailed balance** equation:

\[ \pi(x) Pr(y \mid x) = \pi(y) Pr(x \mid y) \]
METROPOLIS-HASTINGS SAMPLING

1. Initialise Parameters
2. Propose New Value
3. Evaluate Acceptance
   - Reject
     - Revert to Current Value
   - Accept
     - Replace with New Value
Gibbs Sampler
Metropolis-Hastings
Adaptive Metropolis
Hit-and-Run Algorithm
Slice Sampler
This document contains code samples in R and a visual representation of a WinBUGS interface. The code includes functions for specifying and monitoring the convergence of Bayesian models. The interface shows node statistics, including means, standard deviations, and quantiles, as well as sample sizes. The code and interface are used for statistical modeling and analysis.
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)
}
## Trap!

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cumulative</td>
<td>REAL</td>
</tr>
<tr>
<td>mu</td>
<td>REAL</td>
</tr>
<tr>
<td>tau</td>
<td>REAL</td>
</tr>
<tr>
<td>x</td>
<td>REAL</td>
</tr>
<tr>
<td>GraphNormalNode</td>
<td>REAL</td>
</tr>
<tr>
<td>logNorm</td>
<td>REAL</td>
</tr>
<tr>
<td>logTau</td>
<td>REAL</td>
</tr>
<tr>
<td>node</td>
<td>GraphNormalNode</td>
</tr>
<tr>
<td>GraphNormalNode</td>
<td>REAL</td>
</tr>
<tr>
<td>GraphCensoringTruncationLimitLT</td>
<td>REAL</td>
</tr>
<tr>
<td>p</td>
<td>GraphNodesNode</td>
</tr>
<tr>
<td>GraphNormalNode</td>
<td>REAL</td>
</tr>
<tr>
<td>GraphStochasticNode</td>
<td>REAL</td>
</tr>
</tbody>
</table>

### GraphNormalNode
- **logNorm** REAL
- **logTau** REAL
- **node** GraphNormalNode
- **x** REAL

### GraphStochasticNode
- **child** GraphStochasticNode
- **likelihood** GraphStochasticList
- **log** REAL

### UpdaterMode
- **chain** INTEGER
- **corl** REAL
- **h** REAL
- **k** INTEGER
- **logRand** REAL
- **alpha** INTEGER
- **overRelax** BOOLEAN
- **prior** GraphStochasticNode
- **rand** INTEGER
- **res** SET
- **updater** UpdaterDiscreteSliceStdUpdater

### UpdaterActionsSample
- **chain** INTEGER
- **current** UpdaterActionsUpdaterList
- **overRelax** BOOLEAN
- **res** SET
- **updater** UpdaterDiscreteSliceStdUpdater

### BugInterfaceUpdaterMode
- **chain** INTEGER
- **error** ARRAY 120 OF CHAR
- **i** INTEGER
- **j** INTEGER
- **k** INTEGER
- **l** INTEGER
OPENBUGS
NUMBER OF DEVELOPERS: 0.3
“OPENBUGS IS OPEN SOURCE ONLY IN THE READ-ONLY SENSE.”

Andrew Thomas
Atrocities include:

» requires BlackBox Component Builder (Windows only)
» no debugger
» source code is not plain text
» abandoned by Oberon Microsystems
“A PLATFORM FOR EXPERIMENTING WITH IDEAS IN BAYESIAN MODELING”
ITS JUST WINBUGS!

» uses variant of BUGS language

» interpreter creates virtual graphical model
IT'S NOT WINBUGS!

» no graphical front-end

» no output processing

» cross-platform

» extensible: C/C++/Fortran
proc mcmc data=stagnant outpost=postout seed=24860 ntu=1000 nmc=20000;
ods select PostSummaries;
ods output PostSummaries=ds;

array beta[2];
parms alpha cp beta1 beta2;
parms s2;

prior cp ~ unif(-1.3, 1.1);
prior s2 ~ uniform(0, 5);
prior alpha beta: ~ normal(0, v = 1e6);

j = 1 + (x >= cp);
mu = alpha + beta[j] * (x - cp);
model y ~ normal(mu, var=s2);
run;
```
beta = pm.Normal("beta", 0, 0.0001)
alpha = pm.Normal("alpha", 0, 0.0001)

@prulet 1.1
def mean(X=x, alpha=alpha, beta=beta):
    return alpha + beta * X

obs = pm.Normal("obs", mean, prec, value=Y, observed=True)
mcmc = pm.MCMC([obs, beta, alpha, std, prec])

mcmc.sample(100000, 80000)
mcpplot(mcmc)

[*********************100%*********************] 100000 of 100000 complete
Plotting beta
Plotting alpha
```
stochastic

alpha

beta

a

b

theta

p

deaths

deterministic

data stochastic
>>> theta = Exponential('theta', 2.)
>>> theta
<pymc.distributions.Exponential 'theta' at 0x10f9e2790>
>>>
```python
>>> theta = Exponential('theta', 2.)
>>> theta
<pymc.distributions.Exponential 'theta' at 0x10f9e2790>
>>> theta.value
array(0.3848267102336743)
```
>>> theta = Exponential('theta', 2.)
>>> theta
<pymc.distributions.Exponential 'theta' at 0x10f9e2790>
>>> theta.value
array(0.3848267102336743)
>>> theta.logp
-0.07650623990740335
>>>
```python
>>> theta = Exponential('theta', 2.)
>>> theta
<pymc.distributions.Exponential 'theta' at 0x10f9e2790>
>>> theta.value
array(0.3848267102336743)
>>> theta.logp
-0.07650623990740335
>>> theta.random()
array(0.4290156681214033)
```
```python
def sample_theta():
    theta = Exponential('theta', 2.)
    theta.value
    print(theta.logp)
    theta.random()
    print(theta.value)

to_run = sample_theta()
```
>>> theta = Exponential('theta', 2.)
>>> theta
<pymc.distributions.Exponential 'theta' at 0x10f9e2790>
>>> theta.value
array(0.3848267102336743)
>>> theta.logp
-0.07650623990740335
>>> theta.random()
array(0.4290156681214033)
>>> theta.value
array(0.4290156681214033)
>>> theta.logp
-0.1648841556828613
@observed
def survival(value=t, lam=theta, f=failure):
    """Exponential survival likelihood, accounting for censoring""
    return sum(f * log(lam) - lam * value)
from pymc import Normal, Lambda, observed
from melanoma_data import *

# Convert censoring indicators to indicators for failure event
failure = (censored==0).astype(int)

# Intercept for survival rate
beta0 = Normal('beta0', mu=0.0, tau=0.0001, value=0.0)
# Treatment effect
beta1 = Normal('beta1', mu=0.0, tau=0.0001, value=0.0)

# Survival rates
lam = Lambda('lam', lambda b0=beta0, b1=beta1, t=treat: exp(b0 + b1*t))

@observed
def survival(value=t, lam=lam, f=failure):
    """Exponential survival likelihood, accounting for censoring""
    return sum(f*log(lam) - lam*value)
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)
The Hamiltonian is defined as the sum of potential energy $E(s)$ and kinetic energy $K(\phi)$

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

\[
\frac{d\phi_i}{dt} = -\frac{\partial H}{\partial s_i} = -\frac{\partial E}{\partial s_i}
\]
LEAPFROGGING

\[ \phi_i(t + \epsilon/2) = \phi_i(t) - \frac{\epsilon}{2} \frac{\partial}{\partial s_i} E(s(t)) \]

\[ s_i(t + \epsilon) = s_i(t) + \epsilon \phi_i(t + \epsilon/2) \]

\[ \phi_i(t + \epsilon) = \phi_i(t + \epsilon/2) - \frac{\epsilon}{2} \frac{\partial}{\partial s_i} E(s(t + \epsilon)) \]
1. Sample a new velocity from univariate Gaussian

2. Perform n leapfrog steps to obtain new state $\chi'$

3. Perform accept/reject move of $\chi'$
with pm.Model() as model:
    pm.glm.glm('y ~ x', data)
    step = pm.Metropolis()
    iter_sample = pm.iter_sample(samples+1000, step)

animation.FuncAnimation(fig, animate, init_func=init,
                        frames=samples, interval=5, blit=True)
with pm.Model() as model:
    pm.glm.glm('y ~ x', data)
    step = pm.NUTS()
    iter_sample = pm.iter_sample(samples+1000, step)

animation.FuncAnimation(fig, animate, init_func=init,
                        frames=samples, interval=5, blit=True)

VIDEO COURTESY THOMAS WIECKI (@TWIECKI)
Hamiltonian MC requires tuning of two parameters:

**TRAJECTORY LENGTH**

» too small $\Rightarrow$ slow mixing (random walk)

» too large $\Rightarrow$ u-turns

**LEAPFROG STEP SIZE**

» too small $\Rightarrow$ inefficient simulation

» too large $\Rightarrow$ inaccurate simulation
GO NUTS WITH THE NO U-TURN SAMPLER
HOFFMANN AND GELMAN (2011)
NUTS uses a recursive algorithm to build a set of likely candidate points that spans a wide swath of the target distribution.

It stops automatically when it starts to double back and retrace its steps.
Doubling process builds a balanced binary tree whose leaf nodes correspond to position-momentum states.

Doubling is halted when the subtrajectory from the leftmost to the rightmost nodes of any balanced subtree of the overall binary tree starts to double back on itself.
NUTS sampling tends to be more independent than manually-tuned HMC sampling.
Relative Number of Births

Trend
- Slow trend
- Fast non-periodic component
- Mean

Day of week effect
- 1972
- 1976
- 1980
- 1984
- 1988

Seasonal effect
- 1972
- 1976
- 1980
- 1984
- 1988

Day of year effect
- Valentine’s day
- Memorial day
- Labor day
- Thanksgiving

Andrew Gelman, John B. Carlin, Hal S. Stern,
David B. Dunson, Aki Vehtari, and Donald B. Rubin
STAN IS ...
» compiled
» statically typed
» imperative
» BSD licenced
FOUR FLAVORS OF STAN

1. CmdStan
2. RStan
3. PyStan
4. MStan
Hierarchical model example

data {
    int<lower=0> J; // number of schools
    real y[J]; // estimated treatment effects
    real<lower=0> sigma[J]; // s.e. of effect estimates
}

parameters {
    real mu;
    real<lower=0> tau;
    real eta[J];
}
transformed parameters {
  real theta[J];
  for (j in 1:J)
    theta[j] <- mu + tau * eta[j];
}

model {
  eta ~ normal(0, 1);
  y ~ normal(theta, sigma);
}
Calculating Gradients in Theano

```python
>>> from theano import function, tensor as T
>>> x = T.dmatrix('x')
>>> s = T.sum(1 / (1 + T.exp(-x)))
>>> gs = T.grad(s, x)
>>> dlogistic = function([x], gs)
>>> dlogistic([[3, -1],[0, 2]])

array([[ 0.04517666,  0.19661193],
       [ 0.25      ,  0.10499359]])
```
WHAT'S NEW?

» HMC/NUTS

» Slice sampler

» Parallel chains

» Simplified code base

» R-style formulae for GLM
from pymc import *
from pandas import read_csv
htwt_data = read_csv('data/HtWt.csv')

with Model() as model_htwt:
    glm.glm('male ~ height + weight', htwt_data, family=glm.families.Binomial())
    trace_htwt = sample(1000, NUTS())
QUESTION: I HATE MCMC. WHAT ELSE CAN I DO?
APPROXIMATE BAYESIAN COMPUTING
LIKELIHOOD-FREE COMPUTATION
BECAUSE LIKELIHOODS DON'T CALCULATE THEMSELVES, YOU KNOW.
1. Sample parameters $\theta$ from priors $P(\theta)$.

2. Simulate data $x$ from sampling distributions $x \sim P(x|\theta)$.

3. If distance $x=y$, keep $\theta$ as sample from posterior.

4. Repeat steps 1-3 until desired number of samples is obtained.
**ABC ALGORITHM**

1. Sample parameters $\theta$ from priors $P(\theta)$.

2. Simulate data $x$ from sampling distributions $x \sim P(x|\theta)$.

3. If distance $\rho(x, y) < \epsilon$, keep $\theta$ as sample from posterior.

4. Repeat steps 1-3 until desired number of samples is obtained.
INLA
INTEGRATED NESTED LAPLACE APPROXIMATIONS
RUE AND MARTINO (2009)
Posterior Marginals
Latent Variables

\[ \pi(x_i \mid y) = \int \pi(x_i \mid \theta, y) \pi(\theta \mid y) d\theta \]

\[ \tilde{\pi}(x_i \mid y) = \int \tilde{\pi}(x_i \mid \theta, y) \tilde{\pi}(\theta \mid y) d\theta \]
Posterior Marginals

Hyperparameters

$$\pi(\theta_j | y) = \int \pi(\theta | y) d\theta_{-j}$$

$$\tilde{\pi}(\theta_j | y) = \int \tilde{\pi}(\theta | y) d\theta_{-j}$$
1. Approximate the posterior marginal of $\theta$ using the Laplace approximation

2. Compute the Laplace approximation of $\pi(x|y, \theta)$, for selected values of $\theta$, in order to improve on the Gaussian approximation

3. Combine outputs of (1) and (2) using numerical integration
\[ \tilde{\pi}(\theta | y) \propto \frac{\pi(x, \theta, y)}{\tilde{\pi}_G(x | \theta, y)} \bigg| x = x^*(\theta) \]
\[ \tilde{\pi}_{LA}(x_i | \theta, y) \propto \frac{\pi(x, \theta, y)}{\tilde{\pi}_G(x_{-i} | x_i, \theta, y)} \bigg|_{x_{-i} = x^*_{-i}(x_i, \theta)} \]
data(Surg)

formula = r ~ f(hospital,model="iid",param=c(0.001,0.001))

mod.surg = inla(formula,data=Surg,family="binomial",Ntrials=n)

# improved estimation of the hyperparameters
h.surg = inla.hyperpar(mod.surg)

[r-inla.org]
The Future
“The best way to predict the future is to invent it.”

Alan Kay