Scientific Computing for ML and Supercloud

Jonathan Huggins

Scientific Computing for Machine Learning[1]

- Challenges
  - Algorithms are stochastic
  - Correct algorithms can perform badly
  - What counts as good performance?
  - ML algorithms can be robust to bugs!
    - Fails on large datasets, in high dimensions, etc.
  - *Unforeseen interactions between ML algorithms*

[1]Based on Grosse and Duvenaud (2014). "Testing MCMC code." http://arxiv.org/abs/1412.5218

Scientific Computing for Machine Learning[1]

- Best practices
  - modular code
  - separate logically distinctive components
    - e.g., optimizer + function being optimized
    - e.g., sampler + model
  - testing
    - unit tests: correctness of small piece of code
    - integration tests: correctness of system

[1]Based on Grosse and Duvenaud (2014). "Testing MCMC code." http://arxiv.org/abs/1412.5218

Example: Gibbs sampler for a mixture of Gaussians model[1]

$$\pi \sim \mathrm{Dirichlet}(\alpha)$$

$$\sigma_\mu^2 \sim \mathrm{InverseGamma}(a_\mu, b_\mu)$$

$$\sigma_n^2 \sim \mathrm{InverseGamma}(a_n, b_n)$$

$$z_i \,|\, \pi \sim \mathrm{Multinomial}(\pi)$$

$$\mu_{kj} \,|\, \sigma_\mu^2 \sim \mathrm{Normal}(0, \sigma_\mu^2)$$

$$x_{ij} \,|\, z_i, \mu_{z_i,j}, \sigma_n^2 \sim \mathrm{Normal}(\mu_{z_i,j}, \sigma_n^2)$$

[1]Based on Grosse and Duvenaud (2014). "Testing MCMC code." http://arxiv.org/abs/1412.5218

Gibbs sampling

- Model is p(x, y, z)

- Start with state (x, y, z), then
  x’ ~ p(. | y, z)
  y’ ~ p(. | x’, z)
  z’ ~ p(. | x’, y’)

- Set (x, y, z) ← (x’, y’, z’)

- Repeat

[1]Based on Grosse and Duvenaud (2014). “Testing MCMC code.” http://arxiv.org/abs/1412.5218

Writing MCMC code

- For MCMC, split code into
  - Distributions
  - Model
  - Sampler
- Modularity makes testing and reuse easier

<show MoG code>

$$
\begin{aligned}
\pi &\sim \text{Dirichlet}(\alpha) \\
\sigma_\mu^2 &\sim \text{InverseGamma}(a_\mu, b_\mu) \\
\sigma_n^2 &\sim \text{InverseGamma}(a_n, b_n) \\
z_i \,|\, \pi &\sim \text{Multinomial}(\pi) \\
\mu_{kj} \,|\, \sigma_\mu^2 &\sim \text{Normal}(0, \sigma_\mu^2) \\
x_{ij} \,|\, z_i, \mu_{z_i,j}, \sigma_n^2 &\sim \text{Normal}(\mu_{z_i,j}, \sigma_n^2)
\end{aligned}
$$

Unit Testing

- Ideally, write the tests before writing the code
- Modular code makes writing unit tests easier
- Write and test "naive" implementation before adding an optimized implementation
- Trick for testing conditional probabilities: check that

$$p(x \mid z)/p(x' \mid z) = p(x, z)/p(x', z)$$

holds for random x, x', z (normalization terms will cancel)

```python
class Model:
    ...

    def joint_log_p(self, state, X):
        return DirichletDistribution(self.alpha * np.ones(self.K)).log_p(state.pi) + \
               MultinomialDistribution.from_probabilities(state.pi).log_p(state.z).sum() + \
               self.sigma_sq_mu_prior.log_p(state.sigma_sq_mu) + \
               self.sigma_sq_n_prior.log_p(state.sigma_sq_n) + \
               GaussianDistribution(0., state.sigma_sq_mu).log_p(state.mu).sum() + \
               GaussianDistribution(state.mu[state.z, :], state.sigma_sq_n).log_p(X).sum()
```

```python
def test_cond_mu():
    model = random_model()
    state, X = model.forward_sample(N, D)
    new_state = copy.deepcopy(state)
    new_state.mu = np.random.normal(size=(K, D))
    cond = model.cond_mu(state, X)
    assert np.allclose(cond.log_p(new_state.mu).sum() - cond.log_p(state.mu).sum(),
                       model.joint_log_p(new_state, X) - model.joint_log_p(state, X))
```

Unit Testing in Python

- In python, `nose` makes testing very easy
    - name test files `test_*` and test functions `test_*`
- also very useful: `numpy.testing`
    - when a test fails, provides useful debugging info

Integration Testing

- Geweke test:

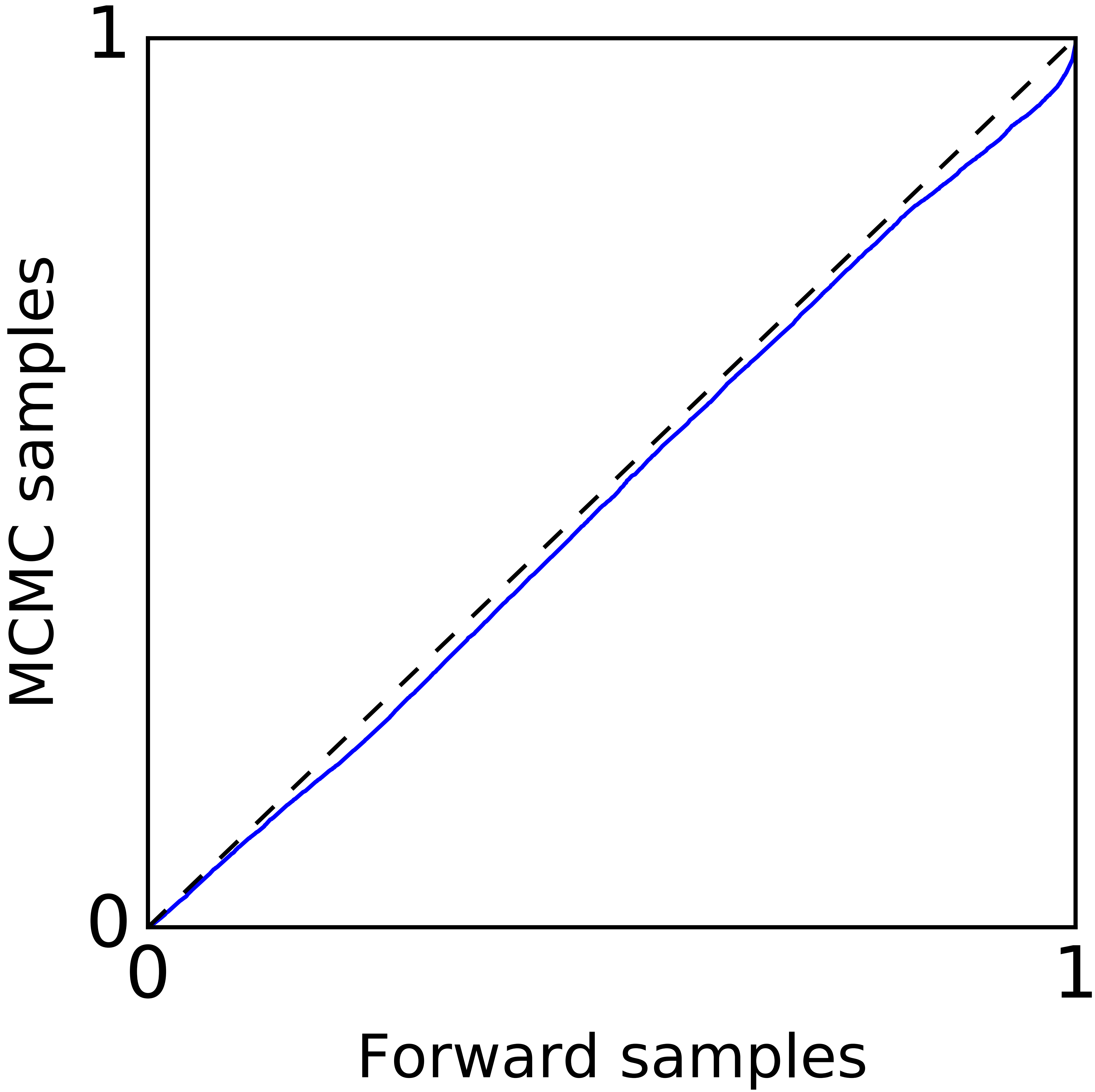
  1. Sample iid from full model (data and latents)

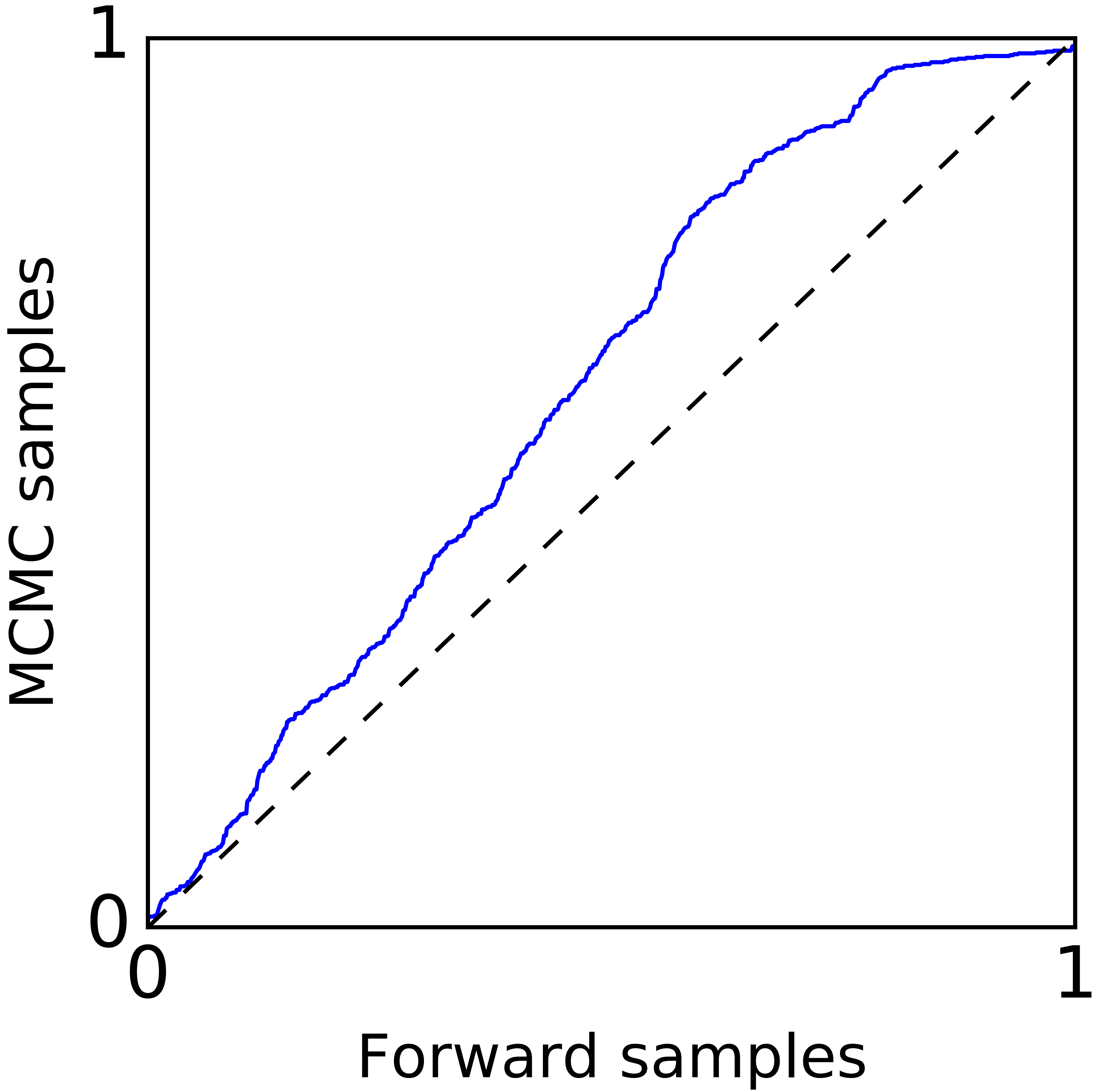
  2. Iterate between (a) sampling new latents using MCMC algorithm and (b) resampling data from forward model

  3. Compare samples from steps 1 + 2 using P-P plot:

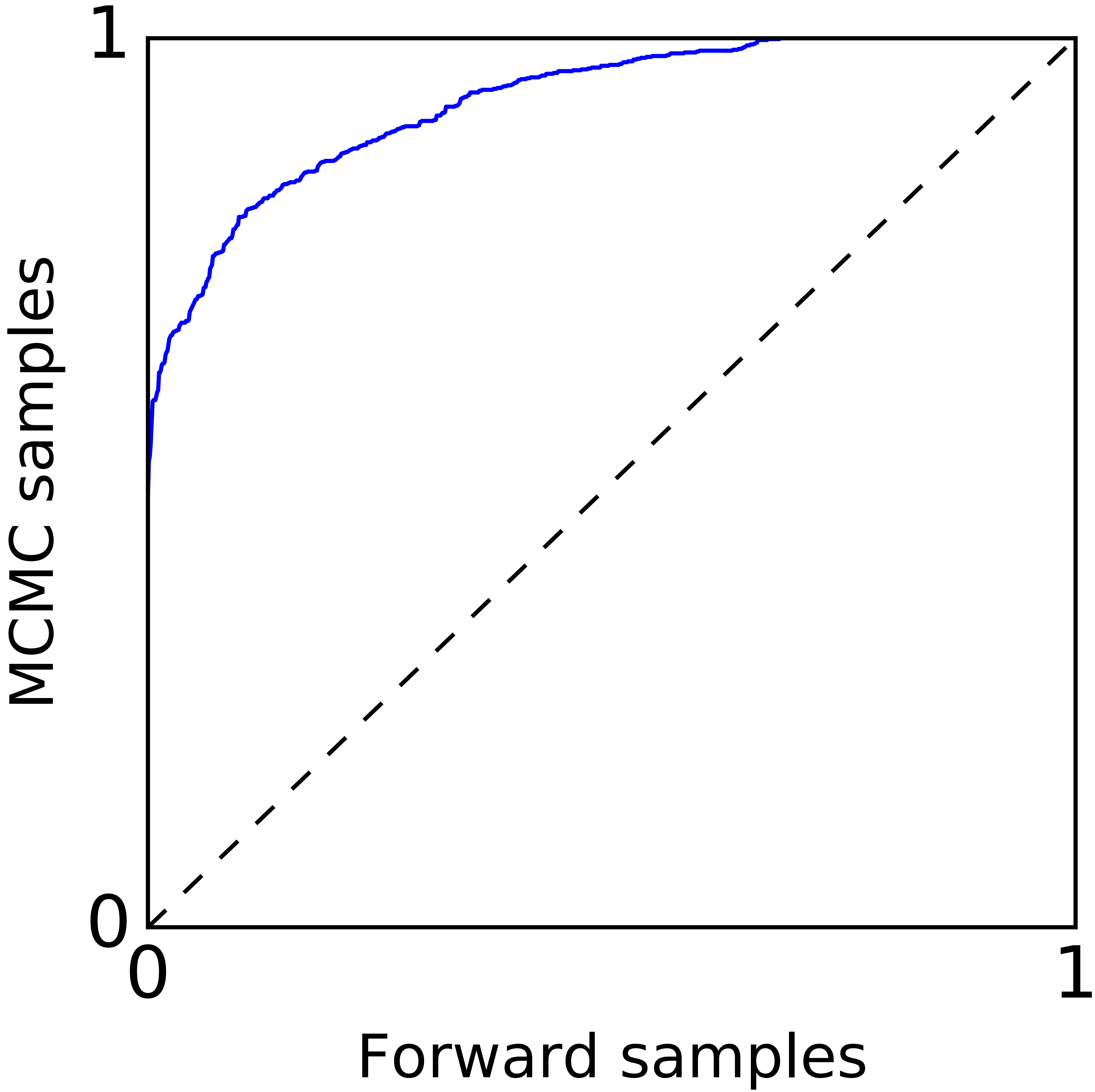
     - for CDFs F and G, plot (F(z), G(z)) for z in (-inf, inf)

- Geweke test can fail because of bug or poor mixing


1
0
0
1
MCMC samples
Forward samples



1
0
0
1
MCMC samples
Forward samples


1
0
0
1
MCMC samples
Forward samples

Using Supercloud

- Supercloud is a high-performance cluster available through Lincoln Labs
- ssh <username>@txe1-login.mit.edu
- LLsub <job-script> -o <log-file> -J <job-name>
- See http://groups.csail.mit.edu/broderick/wiki/index.php?title=Supercloud
- Please add whatever you learn to the wiki!

Future topics…

- Using Cython (easy integration of C code with Python)
- Gelman–Rubin diagnostics for debugging MCMC
- Coding for GPUs (if anyone knows how)
- Autodiff