Stochastic Models

Walt Pohl

Universität Zürich
Quantitative Business Administration

September 9, 2011

Introduction to MCMC

Markov chain Monte Carlo (MCMC) is the most sophisticated method for simulating draws from a random variable.

The only thing we need to know is the probability density function, up to a constant factor.

Surprisingly indirect.

Markov Chain

A *Markov chain* is sequence of random variables, x_i so that probability distribution of the next variable depends (only) on the current value.

Example: A random walk,

$$x_{i+1} = x_i + \epsilon_i$$

Note that in this literature the Markov chains have continuous state spaces.

Stationary Distribution

Under some conditions, long sequences of draws from a Markov chain converge to draws from a single distribution, the *stationary distribution*.

For example, long draws from an AR(1) process

$$x_{i+1} = \rho x_i + \epsilon_i$$

converge to draws from $\mathit{Normal}(0, \sigma_\epsilon^2/(1-\rho^2))$, if $-1 < \rho < 1$.

Delicate Balance Condition

So if we can find a Markov chain with a specified stationary distribution, we can use the chain to simulate draws from the that distribution.

One condition that ensures it is the *delicate balance condition*,

$$T(x_{i-1}|x_i)p(x_i) = T(x_i|x_{i-1})p(x_{i-1})$$

where p is the pdf of the stationary distribution, while T is the conditional distribution of the Markov chain.

Delicate Balance Condition, cont'd

This family has another convenient property: it imposes no condition on $T(x_i|x_i)$.

We exploit this.

Outline of MCMC

Let x_i be the current draw.

- We draw x^* from an arbitrary Markov chain, with conditional density $q(x^*|x_i)$.
- We turn it into the desired chain by changing how often we stay in the current state.
- We do this by performing additional draws from a 0-1 random variable. If 1, we *accept* x^* as the next draw, x_{i+1}. Otherwise, we *reject*, and use the previous draw, x_i as the next draw.

Acceptance Probability

It's not hard to show that the following choice of acceptance probability does the trick.

$$\min\left(1, \frac{p(x^*)q(x_i|x^*)}{p(x_i)q(x^*|x_i)}\right)$$

This is the Metropolis-Hastings algorithm.

Metropolis Algorithm

If we specialize to a symmetric random walk,

$$q(x'|x) = q(x|x'),$$

this simplifies to

$$\min\left(1, \frac{p(x^*)}{p(x_i)}\right).$$

This is the Metropolis algorithm.

Why MCMC?

So why would you need such an elaborate Monte Carlo procedure?

The main application: Bayesian estimation in statistics.

The output of Bayesian estimation is the pdf of a probability distribution.

Bayesian estimation: The Model

In Bayesian estimation, you start with a completely specified model (depending on parameters), including distributional assumptions for all variables.

Example:

$$x_i = \mu + \epsilon_i,$$

Must specify distribution for ϵ_i, such as

$$\epsilon_i \sim Normal(0, \sigma^2).$$

So this model depends on μ and σ^2. The model tells us a density, $p(\epsilon_1, \ldots, \epsilon_n | \mu, \sigma^2)$.

Bayesian estimation: The Prior

You assume that *the parameters themselves are random*, and pick a probability distribution for them. This is the *prior*.

Example:

- $\mu \sim Normal(\mu', \sigma')$
- $1/\sigma^2 \sim \Gamma(k, \theta)$

The prior gives us a density $p(\mu, \sigma^2)$.

What about μ', σ', k, and θ? These are called *hyperparameters*, but they're just constants we fit at the beginning of the analysis.

Bayesian estimation: The Posterior

We compute the *posterior* for the data $\epsilon_1, \ldots, \epsilon_n$.

$$p(\mu, \sigma^2 | \epsilon_1, \ldots, \epsilon_n) = \frac{p(\epsilon_1, \ldots, \epsilon_n | \mu, \sigma^2) p(\mu, \sigma^2)}{\int p(\epsilon_1, \ldots, \epsilon_n | \mu, \sigma^2) p(\mu, \sigma^2) d\mu d\sigma^2}$$

The posterior tells us the probability of seeing the parameters, given the data. This clearly depends on the choice of prior.

Choosing a Prior

So where does the prior come from?

- In theory: the prior should represent your subjective probability assessment of the outcomes.
- In practice: there are customary choices of priors, such as in the previous example.

Asymptotically, the posterior converges to the correct point estimate – it's a *consistent estimator*. So we can think of Bayesian posteriors as just another class of estimators.

Summarizing the Posterior

To summarize the posterior, one usually reports:

- Mean of posterior – point estimate.
- Standard deviation of posterior – estimate of precision (analogous to the standard error).

How do we compute these? We know the pdf of the posterior up to a constant factor,

$$p(\epsilon_1, \ldots, \epsilon_n | \mu, \sigma^2) p(\mu, \sigma^2),$$

so we can use Monte Carlo simulation of the posterior using MCMC.

Structural Estimation

What do we want to use it for? Structural estimation.

- Take a fully-parametrized model such as the RBC model.
- The model makes predictions for endogenous variables as a function of exogenous variables.
- Choose data to represent the variables, and choose parameters that best match the data.
- Alternative to calibration.

Bayesian Structural Estimation

Bayesian estimation using MCMC can be used for a relatively simple approach to structural estimation.

- Easy part, thanks to MCMC: draw parameters from the posterior.
- Hard part, solve the model using the parameters.

Advantages of Bayesian Structural Estimation

Since we have a complete model, we have the likelihood, and could use maximum likelihood estimation. The advantages of the Bayesian approach is:

- The likelihood may not be concave, so a local solver may have find a local maximum. MCMC explores a large part of parameter space.
- Smooth solvers require derivatives, so the model solution must approximate the derivatives well.

The disadvantage of MCMC is that you usually have to do a gigantic number of draws, say 10,000,000. So you have to solve your model 10,000,000 times.

Endowment Economy

Example application: asset prices in an endowment economy. How well do aggregate consumption and dividend data explain stock and bond prices?

The question is studied in a standard framework:

- A single representative investor optimizing over time.
- Investor has CRRA preferences.
- Model is tested using US consumption and price data: 1889-2004.

Choice Variables

Each period, representative investor has wealth W_t, and chooses between:

- Consumption, quantity C_t, price 1,
- Buying a risky asset, quantity S_t, price P_t,
- Buying a risk free bond, quantity B_t, price P_t^f.

Budget constraint:

$$W_t = C_t + S_t P_t + B_t P_t^f.$$

In equilibrium $S_t = S$, $B_t = 0$.

Wealth

Each period, representative investor receives:

- Endowment income, Q_t.
- Dividends from risky asset, $D_t S_{t-1}$.
- Bond payment, B_{t-1}.

Wealth is

$$W_t = Q_t + (P_t + D_t)\, S_{t-1} + B_{t-1}$$

In equilibrium, $W_t = Q_t + P_t + SD_t$.

Agent Objective

Agent objective is a dynamic programming problem:

$$V(W_t) = \max u(C_t) + \beta E_t V(W_{t+1})$$

where

$$u(x) = \frac{x^{1-\gamma}}{1-\gamma}.$$

Solving the Model

Suppose C_t and D_t are log-AR(1). Then you can use one of many numerical methods to find:

- $P(D_t, C_t)$
- $P_f(D_t, C_t)$

I used Galerkin.

Note that this predicts prices *exactly* as a function of the state variables. Unless we have a model that fits exactly, we can't fit this model.

What To Estimate?

We assume that prices are observed *with error*.

We have a model for four series:

- $\log C_{t+1} = (1-\rho_c)\mu_c + \rho_c \log C_t + \epsilon_{ct}$,
- $\log D_{t+1} = (1-\rho_d)\mu_d + \rho_d \log D_t + \epsilon_{dt}$,
- $\log P_t = \log P(D_t, C_t) + \epsilon_{pt}$,
- $\log P_t^f = \log P^f(D_t, C_t) + \epsilon_{pft}$.

All innovations are normally distributed.

Data

I use:

- C_t is annual US consumption.
- P_t and D_t are annual prices and dividends for the S&P.
- P_t^f is the annualized 3-month interest rate on US debt.

The first three grow over time, so I detrend them.

Parameters

The model depends on a bunch of parameters:

- Time-series parameters, ϵ, μ, and ρ.
- Utility parameters: β, γ.

For a prior, I let the parameters be uniformly distributed over their domain of definition.

I simulate the posterior using the Metropolis algorithm with normal draws.

My Experience with MCMC

- I spent a month implementing the Galerkin solution step (in C).
- I spent less than a day implementing the MCMC.
- The Galerkin step takes about a second.
- To run it 10,000,000 times took about a month.