Monte Carlo Integration

Anthony Stephenson

12/1/2020

Contents

Hamiltonian Monte Carlo (HMC)

Hamiltonian Monte Carlo is a method of sampling from a posterior distribution whose normalisation factor is intractable, such that we can calculate unnormalised probabilities and the unnormalised gradient of the posterior. An auxiliary variable, the momentum, is introduced and we describe the system by its Hamiltonian. We hope that by incorporating the gradient we can make the sampling procedure more efficient, by guiding the Markov chain along regions of high probability mass, rather than via a random walk as in standard MCMC schemes.

Define a Hamiltonian

$$\begin{aligned} H(\rho, \theta) &= T(\rho \mid \theta) + V(\theta)) \\ &= -\log p(\rho \mid \theta) - \log p(\theta) \end{aligned}$$

where T is the kinetic energy of the system and V is the potential and, in the parlance of Hamiltonian dynamics, θ is the position and ρ is the momentum of the system. If $p(\rho \mid \theta)$ is a Gaussian distribution, then this corresponds to the usual definition of kinetic energy as $\frac{1}{2}p^T M^{-1} p$ (where M is the "mass matrix") using a Boltzmann-type distribution of $p = \frac{1}{Z}\mathrm{e}^{-H/kT}$ where we choose the units of temperature T s.t. $kT = 1$.

To make use of this method to sample parameters we must be able to both evaluate the potential and the gradient of the potential at all sample points.

Hamiltonian dynamics

Hamiltonian dynamics describes the evolution of a dynamical system via a series of related differential equations of the position (q) and momentum (p) of some object. It is intimately related to Lagrangian dynamics (via a Legendre transform - $H = \sum_i p_i \dot{q}_i - L$), which can be derived from Hamilton's principle, which states that the motion of a system from some state A to a new state B is that which minimises the action, $S = \int_A^B L dt$. In the Hamiltonian formulation, the variables q and p are effectively interchangeable in terms of their interpretation and hence can be applied quite generally to problems of the appropriate form.

Key properties:

- Time reversibility i.e. we can run the dynamics forwards or backwards in time and get consistent results.

- Louiville's theorem states that the volume of phase space is constant. Phase space is the combined space of $(q, p) \in R^{2d}$ the generalised position and momentum coordinates.
- Hamiltonian conservation i.e. $\frac{dH}{dt} = 0$.

Hamilton's equations

Hamilton's equations are derived by combining the Euler-Lagrange $\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) = \frac{\partial L}{\partial q_i}$ equations with the definitions of the Hamiltonian and generalised momentum (as conjugate to the position) $p_i = \frac{\partial L}{\partial \dot{q}_i}$. For the Hamiltonian $H(\rho, \theta)$ defined above we can write the equations as

$$\frac{d\theta}{dt} = +\frac{\partial H}{\partial \rho} = +\frac{\partial T}{\partial \rho}$$
$$\frac{d\rho}{dt} = -\frac{\partial H}{\partial \theta} = -\frac{\partial T}{\partial \theta} - \frac{\partial V}{\partial \theta}$$

We can solve these differential equations to estimate the evolution of the parameters. We do this with a numerical integrator routine, such as the leapfrog algorithm. Although many numerical intergrators exist, we want to pick a *symplectic* integrator that preserves area in phase space. This rules out common algorithms, such as Euler's method or Runge-Kutta methods.

Implementation

Here is a manual implementation of HMC using a leapfrog numerical integrator. To account for bias introduced in the numerical integration procedure, the transitions are not carried out directly from the integration, but are used as proposals to a Metropolis acceptance scheme.

```
set.seed(123)

# compute MVN density
dmvnorm <- function(x, mu, Sigma) {
  d <- length(mu)
  1/(2 * pi)^(d/2) * 1/sqrt(det(Sigma)) * exp(-0.5 * t(x-mu) %*% (solve(Sigma, (x-mu))))
}

# sample from MVN
mvrnorm <- function(n, mu, Sigma) {
  d <- length(mu)
  # Run eigen decomposition to construct orthogonal matrix U, and diagonal matrix D, s.t. UDU^T = S^-1
  e <- eigen(Sigma)
  U <- e$vectors
  D <- e$values

  # Generate random normal samples of each dimension of the variable in the diagonalised space, y = U^T
  Y <- matrix(0, ncol = d, nrow = n)
  for (i in 1:d) {
    Y[, i] <- rnorm(n) * sqrt(D[i])
  }

  # Generate transformed MVN variables, x
  X <- Y %*% t(U) + mu
  return(X)
}

# numerical integration of Hamilton's equations by the leapfrog algorithm
```

```r
leapfrog <- function(theta0, M, gradV, L, eps=1e-6) {
  rho0 <- t(mvrnorm(1, rep(0, ncol(M)), M))
  rho <- rho0
  theta <- theta0
  for (step in 1:L) {
    rho <- rho - eps/2 * gradV(theta)
    theta <- theta + eps * solve(M, rho)
    rho <- rho  - eps/2 * gradV(theta)
  }
  return(c(rho0, theta0, rho, theta))
}

# accept a proposed state with some probability based on the Hamiltonian
metropolis_accept_step <- function(H, rho, rho_star, theta, theta_star) {
  accept_prob <- min(1, exp(H(rho, theta) - H(rho_star, theta_star)))
  # check sign
  if (runif(1) < accept_prob) {
    return(c(rho_star, theta_star))
  } else {
    return(c(rho, theta))
  }
}

# construct potential function V(theta) from the prior and likelihood functions
construct_potential <- function(prior, likelihood) {
  V <- function(theta) {
    -log(prior(theta)) - log(likelihood(theta))
  }
  return(V)
}

# construct gradient of the potential function, gradV(theta)
construct_grad_potential <- function(grad_prior, grad_lik) {
  gradV <- function(theta) {
    grad_prior(theta) + grad_lik(theta)
  }
  return(gradV)
}

# function to actually carry out the HMC algorithm
hmc <- function(V, theta0, M, L=1, N=100, eps=1e-6) {
  # compute Hamiltonian
  H <- function(rho, theta) {
      d <- length(theta)
    -log(dmvnorm(rho, rep(0, d), M)) + V(theta)
  }
  m <- nrow(M)
  t <- length(theta)
  # generate initial proposal
  proposal <- c(rep(0, m), theta0)
  rec_theta <- matrix(, nrow=N, ncol=t)
  for (step in 1:N) {
    # integrate DEs for L steps to generate new proposal
```

```
    proposal_1 <- leapfrog(proposal[-(1:m)], M, gradV, L, eps=eps)
    # run metropolis accept step to pick whether to keep new proposal
    proposal <- metropolis_accept_step(H, proposal_1[1:m], proposal_1[(m+t+1):(2*m+t)], proposal_1[(m+1
    rec_theta[step,] <- proposal[-(1:m)]
  }
  return(rec_theta)
}
```

As an example pick something simple, e.g. a Bayesian linear model that we can check against. With such a model we can summarise all of the components succinctly, including the posterior:

Prior $p(\theta) = N_\theta(0, \Theta^{-1})$

Likelihood $p(y \mid \theta) = N_y(X\theta, L^{-1})$

Posterior $p(\theta \mid y) = N_\theta(\Sigma X^T L y, \Sigma)$

$-\nabla_\theta \log(p(\theta \mid y)) \ = \Sigma^{-1}(\theta - X^T L y)$

with $\Sigma = (\Theta + X^T L X)^{-1}$.

Now we will attempt to sample from the posterior using HMC.

Provide some more model-specific functions:

```
# Define some model specific functions

# generate a random symmetric positive definite matrix
create_symm_pd_matrix <- function(d) {
  M <- matrix(rnorm(d^2), nrow = d)
  M <- 0.5 * (M + t(M))
  M <- M %*% t(M)
  return(M)
}

# create a likelihood function for MVN
construct_likelihood <- function(y, X, L_inv) {
  likelihood <- function(theta) {
    dmvnorm(y, mu=X %*% theta, Sigma=L_inv)
  }
  return(likelihood)
}

# create a prior function for MVN
construct_prior <- function(m, s) {
  prior <- function(theta) {
    # theta <- matrix(theta, ncol=d, nrow=1)
    dmvnorm(theta, mu=m, Sigma=s)
  }
  return(prior)
}
```

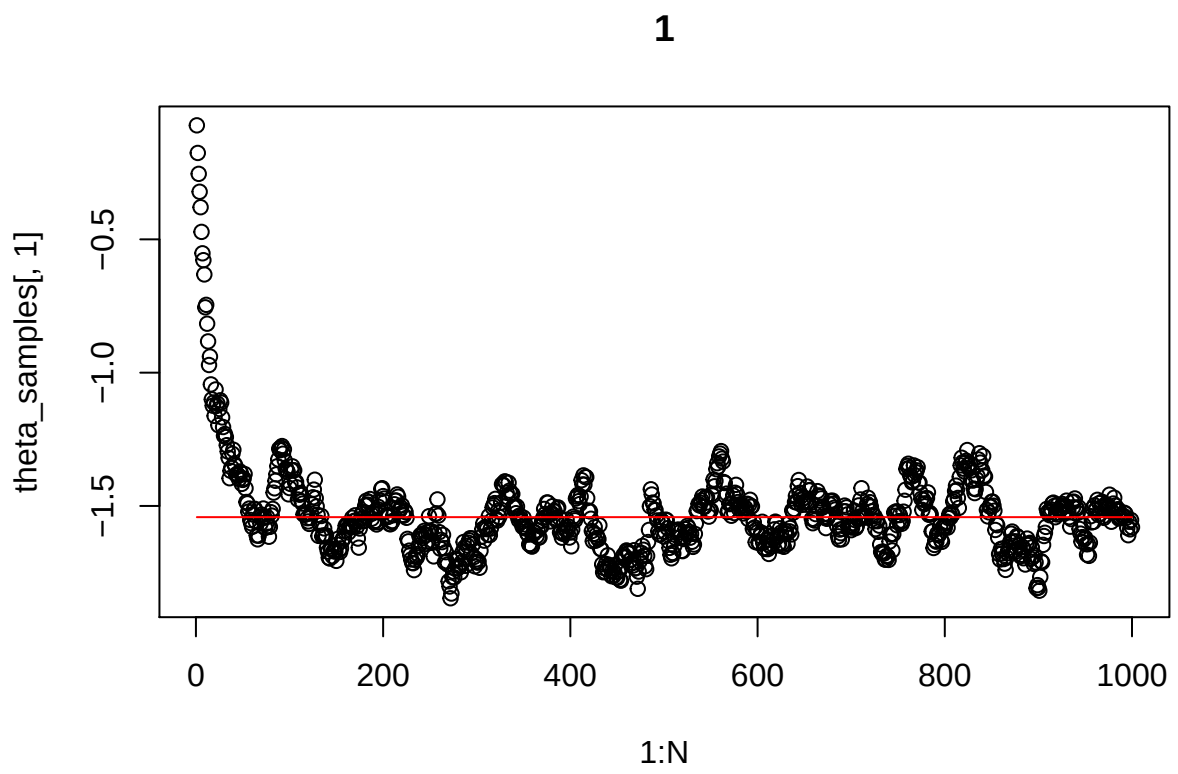
Finally generate some model data and run HMC:

```
# Generate some model data
d <- 2
n <- 100

# Create covariance matrices
```

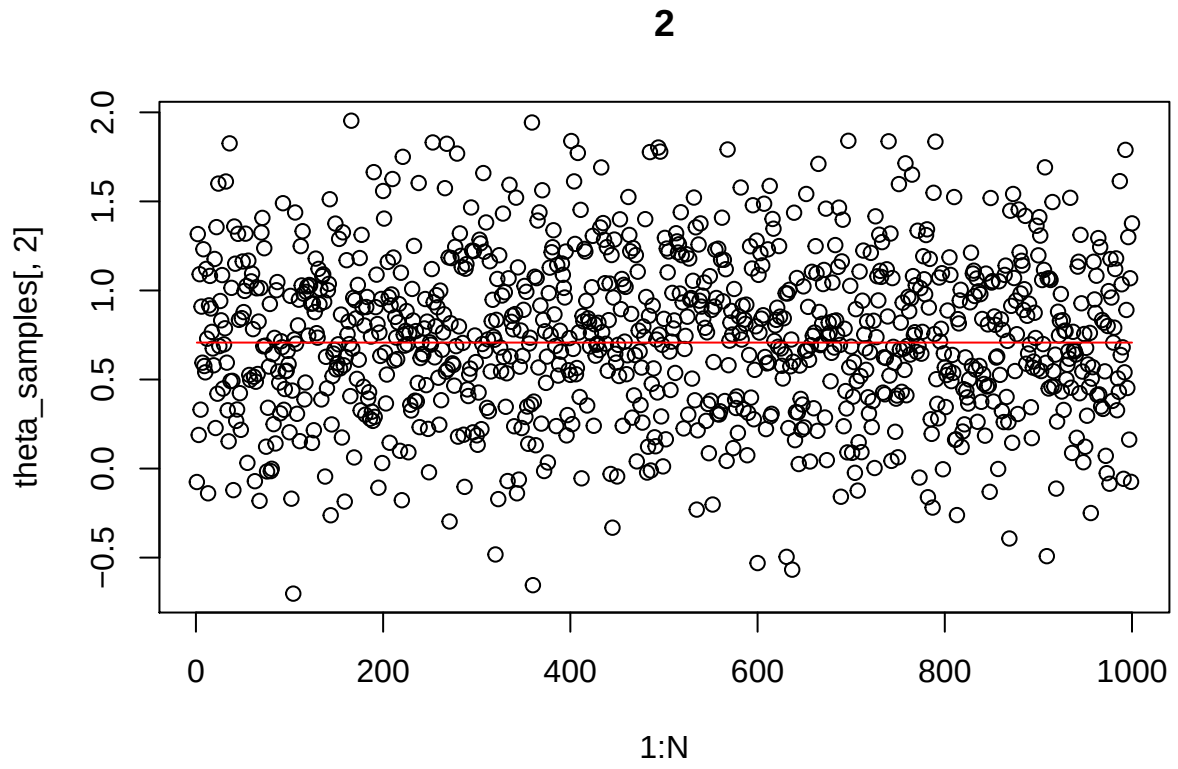
```r
Theta_inv <-  create_symm_pd_matrix(d)
L_inv <- diag(rep(1,n))
covX <- create_symm_pd_matrix(d-1)

# generate coefficients
theta <- t(mvrnorm(1, mu=rep(0, d),  Sigma=Theta_inv))

# Generate data
X <- mvrnorm(n, mu=rep(0, (d-1)), Sigma=covX)
names(X) <- "X"
X <- model.matrix(~1 + X, data=data.frame(X, row.names="X"))

# generate response, based on model
y <- rnorm(n, mean=X %*% theta, sd=L_inv)

# actual posterior covariance
cov_post <- qr.solve(Theta_inv) + t(X) %*% (diag(1/diag(L_inv)) %*% X)

# create prior and likelihood functions
prior <- construct_prior(m=matrix(rep(0,d), nrow=d, ncol=1), s=Theta_inv)
likelihood <- construct_likelihood(y, X, L_inv)

# create potential and gradient of potential (potential=log(posterior))
V <- construct_potential(prior, likelihood)
grad_prior <- function(theta) solve(Theta_inv, theta)
grad_lik <- function(theta) {
  t(X) %*% (1/diag(L_inv) * (X %*% theta)) - t(X) %*% (1/diag(L_inv) * y)
}
gradV <- construct_grad_potential(grad_prior, grad_lik)

# intialise parameters
theta0 <- rep(0,d)
Theta <- qr.solve(Theta_inv)
# use a random symmetric positive definite matrix for the covariance of the momentum
M <- create_symm_pd_matrix(d)

# run hmc
N <- 1000
theta_samples <- hmc(V, theta0, M, L=50, N=N, eps=1e-2)
```

Plot the output samples against the known parameter values:

```r
# output summary
burnin <- 100
theta_hat <- colMeans(theta_samples[-(1:burnin),])
theta_var <- var(theta_samples[-(1:burnin),])

# plot output
y_max <- max(theta[1], max(theta_samples[,1]))
y_min <- min(theta[1], min(theta_samples[,1]))
plot(1:N, theta_samples[,1], ylim=c(y_min, y_max), main="1")
lines(1:N, rep(theta[1], N), col="red")
```




```
y_max <- max(theta[2], max(theta_samples[,2]))
y_min <- min(theta[2], min(theta_samples[,2]))
plot(1:N, theta_samples[,2], ylim=c(y_min, y_max), main="2")
lines(1:N, rep(theta[2], N), col="red")
```



Actual values of parameters: $\theta = (-1.54, 0.707)$

Estimated (average) value of samples: $\hat{\theta} = (-1.56, 0.746)$

Actual posterior covariance: $\Sigma = \begin{bmatrix} 0.00988 & -0.00388 \\ -0.00388 & 0.219 \end{bmatrix}$

Sample covariance: $\hat{\Sigma} = \begin{bmatrix} 0.00968 & -0.00527 \\ -0.00527 & 0.199 \end{bmatrix}$

STAN

The `rstan` package, which wraps the Stan interface, implements HMC in C++ and allows a reasonably straightforward way to build Bayesian models using a dialect similar to BUGS to define models, which gets subsequently translated into C++ code which then runs the HMC algorithm. ("Stan Reference Manual," n.d.)

Below is an example snippet of code to implement the same inference of the linear model as done manually above. The Stan syntax is reasonbly intuitive and generates a detailed output alongside the samples generated.

```
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
## Loading required package: ggplot2
```

```
## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
model_code <- 'data {
  int<lower=0> N;   // number of data items
  int<lower=0> D;   // number of predictors
  matrix[N, D] X;   // predictor matrix
  vector[N] y;      // outcome vector
}
parameters {
  vector[D] theta;       // coefficients for predictors
  vector[D] mu;
  real<lower=0> Theta[D];  //
  real<lower=0> sigma;  // error scale
}
model {
    for (d in 1:D) {
    mu[d] ~ normal(0, 100);
      theta[d] ~ normal(mu[d], Theta[d]);
  }
  y ~ normal(X * theta, sigma);  // likelihood
}'
dat <- list(N = n, D = d, X = X, y = y);
fit <- stan(model_code = model_code, model_name = "example",
            data = dat, iter = 2000, chains = 3, verbose = FALSE, refresh = 0,
            )
```

```
## Warning: There were 2601 divergent transitions after warmup. See
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quant
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess
```

```
print(fit)
```

```
## Inference for Stan model: example.
## 3 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=3000.
##
##                    mean se_mean     sd          2.5%           25%
## theta[1]  -1.560000e+00    0.00   0.02         -1.59 -1.570000e+00
## theta[2]   9.200000e-01    0.01   0.14          0.64  8.300000e-01
## mu[1]      4.940000e+00    4.61 101.64       -193.11 -6.143000e+01
## mu[2]      3.690000e+00    5.05 101.86       -193.05 -6.659000e+01
## Theta[1]  1.069809e+305     NaN    Inf 2657963213.22  1.330041e+78
## Theta[2]  2.302045e+305     NaN    Inf 3712786876.26  1.299347e+78
## sigma      1.700000e-01    0.00   0.01          0.14  1.600000e-01
## lp__       1.262900e+02    0.09   1.72        121.76  1.254500e+02
##                     50%           75%          97.5% n_eff Rhat
## theta[1]  -1.560000e+00 -1.550000e+00 -1.530000e+00   427 1.00
## theta[2]   9.200000e-01  1.020000e+00  1.190000e+00   569 1.00
## mu[1]      7.260000e+00  7.238000e+01  1.997300e+02   486 1.01
## mu[2]      6.540000e+00  7.570000e+01  1.968800e+02   406 1.01
## Theta[1]  2.673133e+149 1.709653e+234 2.079262e+301   NaN  NaN
## Theta[2]  1.387620e+153 4.307963e+232 6.764428e+301   NaN  NaN
## sigma      1.700000e-01  1.700000e-01  1.900000e-01   283 1.01
## lp__       1.266900e+02  1.275600e+02  1.284800e+02   360 1.02
##
## Samples were drawn using NUTS(diag_e) at Fri Jan  8 16:56:48 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Riemannian Hamiltonian Monte Carlo

Riemannian HMC is an extension to the usual HMC algorithm that attempts to exploit information regarding the geometry of the problem by replacing the matrix M that defines the covariance of the momentum with some estimated Riemannian metric tensor. One choice is to compute the Fisher-Rao metric, which is the expectation of the Hessian w.r.t the potential V, i.e. $g_{ij}^{FR} = \mathrm{E}[H(V)]$. Betancourt (2013)

References

Betancourt, Michael. 2013. "A General Metric for Riemannian Manifold Hamiltonian Monte Carlo." *Geometric Science of Information*, 327–34. https://doi.org/10.1007/978-3-642-40020-9_35.

"Stan Reference Manual." n.d. https://mc-stan.org/docs/2_25/reference-manual/hamiltonian-monte-carlo.html.