**HMC Implementation**

1. Introduction
   1. What is HMC: (1p = one paragraph)
      1. HMC has emerged as new method for Bayesian computation
      2. Why HMC is used (high dimensional data)
   2. Where HMC used (1p) current implementation survey (Stan, Pymc)
   3. Why paper (1p)
      1. Why this paper: need for tutorial on how to implement HMC
      2. Use R software only – familiar to statisticians
      3. Emphasis on education over computational efficiency
      4. Assume no background in MCMC (i.e. we start with MH and build)
2. MCMC basic concepts – Metropolis Hastings (careful here don’t take too much time)
   1. Metropolis-Hastings concepts/theory (maybe too much here? My thought though is that HMC will be easier to understand once MH is used).
      1. MCMC – focus on MH (1p)
         1. Mention MH/Gibbs
         2. HMC builds on MH
         3. Start with MH implementation in R
      2. Where MH comes from (1p)
         1. MH algorithm origin: one sentence
         2. Broad: why MH was and is popular
      3. MH algorithm (3p)
         1. Theory: Bayes theorem, likelihood, prior
         2. Posterior: unnormalized. Why MH works
         3. Algorithm specific steps
   2. MH how it works (idea: describe theory, then immediately show the code)
      1. MH R code: show code and then describe

mh <- function(N, paramInit, qPROP, qFUN, pdFUN, nu, ...) {

paramSim <- list()

paramSim[[1]] <- paramInit

accept <- 0

for (j in 2:N) {

u <- runif(1)

paramProposal <- qPROP(paramSim[[j-1]], nu)

lnum <- pdFUN(paramProposal, ...) + qFUN(paramSim[[j-1]], paramProposal, nu)

lden <- pdFUN(paramSim[[j-1]], ...) + qFUN(paramProposal, paramSim[[j-1]], nu)

l.alpha <- pmin(0, lnum - lden)

if (l.alpha > log(u)) {

paramSim[[j]] <- paramProposal

accept <- accept + 1

} else {

paramSim[[j]] <- paramSim[[j-1]]

}

}

list(paramSim = paramSim,

accept = accept)

}

* + - 1. Parameters (N, paramInit, qPROP, qFUN, pdFUN, nu, …) describe each of these and what they are for (1p)
      2. Loop and acceptance ratio (1p)
    1. Linear model example with MH
       1. Likelihood and priors for linear model (1p)
       2. qPROP/qFUN function specification (1p). This is the proposal function for MH – random walk
       3. pdFUN function specific to this linear model (1p). This is the specific unnormalized posterior density. User supplies this
       4. Data example (1p): describe the data used for this example
       5. Implementation (1p): step-by-step tuning of model. Show how different proposals generate different acceptance rates
       6. Trace plots / diagnostics (1p): show traceplots and briefly describe how to diagnose MCMC chain. Describe autocorrelation, stationarity, histogram.
       7. Parameter estimates from MH. Compare to frequentist lm function in R (1p)

1. HMC concepts/theory
   1. Why HMC needed – MH is flexible, but difficult in high-dimensions, can be slow (1p)
   2. HMC differences and similarities to MH (3p)
      1. Gradient of log posterior used to guide proposals (unlike random walk)
      2. Latent variable introduced for each parameter. Mathematical framework from physics – reference Wiley paper
      3. Retains acceptance ratio from MH. Typically higher acceptance than MH. Unnormalized posterior like MH – flexible. However, more parameters in HMC.
   3. HMC basic theory (4p)
      1. Describe Hamiltonian equations. Mention roots in physics
      2. Momentum variable description. P ~ multivariate Normal
      3. Leapfrog algorithm to solve differential equations
      4. MH-style acceptance ratio needed due to approximation (maybe wrap into 3rd paragraph)
   4. HMC algorithm: figure + 1p:
      1. Discuss how this is different from MH. MVN or uniform proposal function in MH compare to MVN momentum + leapfrog
2. HMC implementation – general purpose HMC function in R
   1. HMC R code
3. # leapfrog integrator from Hamiltonian dynamics
4. # theta: parameter of interest
5. # r: momentum variable
6. # epsilon: step size parameter
7. # logDENS: log of joint density of parameter of interest
8. # (log likelihood)
9. # ... additional parameters to pass to logDENS
10. leapfrog <- function(theta\_lf, r, epsilon, logDENS, glPOSTERIOR, y, X, Minv, constrain=FALSE,
11. lastSTEP=FALSE, ...) {
13. # gradient of log posterior for old theta
14. g.ld <- glPOSTERIOR(theta\_lf, y=y, X=X, ...)
16. # first momentum update
17. r.new <- r + epsilon/2\*g.ld
19. # theta update
20. # note diagonal matrix update
21. # theta.new <- theta\_lf + as.numeric(epsilon\*r.new/ diag(M\_mx))
22. theta.new <- theta\_lf + as.numeric(epsilon\* Minv %\*% as.numeric(r.new))
24. # check positive
25. switch\_sign <- constrain & theta.new < 0
26. r.new[switch\_sign] <- -r.new[switch\_sign]
27. theta.new[switch\_sign] <- -theta.new[switch\_sign]
29. # gradient of log posterior for new theta
30. g.ld.new <- glPOSTERIOR(theta.new, y=y, X=X, ...)
32. # if not on last step, second momentum update
33. if (!lastSTEP) {
34. r.new <- r.new + epsilon/2\*g.ld.new
35. }
36. list(theta.new=theta.new,
37. r.new=as.numeric(r.new))
38. }
39. # theta.init: initial values of theta
40. # Nstep: number of steps (called L in paper)
41. # M: number of times to repeat
42. # epsilon: step size
43. # logDENS: log of joint density of parameter of interest
44. # glPOSTERIOR: gradient of log posterior
45. # ...: additional parameters to pass to logDENS
46. hmc <- function(M, thetaInit, epsilon, Nstep, logDENS, glPOSTERIOR, y, X, var.p=NULL, verbose=FALSE, ...) {
47. p <- length(thetaInit) # number of parameters
48. # mass matrix
49. mu.p <- rep(0, p)
50. # unit mass matrix if not provided
51. if (is.null(var.p)) {
52. var.p <- rep(1, p)
53. var.p.mx <- diag(var.p)
54. } else {
55. var.p.mx <- diag(var.p)
56. }
57. # store theta and momentum (usually not of interest)
58. theta <- list()
59. theta[[1]] <- thetaInit
60. r <- list()
61. r[[1]] <- NA
62. accept <- 0
63. for (jj in 2:M) {
64. theta[[jj]] <- theta.new <- theta[[jj-1]]
65. r0 <- MASS::mvrnorm(1, mu.p, var.p.mx)
66. r.new <- r[[jj]] <- r0
67. for (i in 1:Nstep) {
68. lstp <- i == Nstep
69. lf <- leapfrog(theta.new, r.new, epsilon, logDENS, glPOSTERIOR, y, X,
70. var.p.mx, lastSTEP=lstp, ...)
71. #lf <- leapfrog(theta.new, r.new, epsilon, logDENS, y, X, lastSTEP=lstp, a0=a0, b0=b0)
72. theta.new <- lf$theta.new
73. r.new <- lf$r.new
74. }
75. if (verbose) print(jj)
76. # standard metropolis-hastings update
77. u <- runif(1)
78. # use log transform for ratio due to low numbers
79. num <- logDENS(theta.new, y=y, X=X, ...) - 0.5\*(r.new %\*% r.new)
80. den <- logDENS(theta[[jj-1]], y=y, X=X, ...) - 0.5\*(r0 %\*% r0)
81. log.alpha <- pmin(0, num - den)
82. if (log(u) < log.alpha) {
83. theta[[jj]] <- theta.new
84. r[[jj]] <- -r.new
85. accept <- accept + 1
86. } else {
87. theta[[jj]] <- theta[[jj-1]]
88. r[[jj]] <- r[[jj-1]]
89. }
90. }
91. list(theta=theta,
92. r=r,
93. accept=accept)
94. }
95. Describe R code. May rename some of these arguments
    1. Arguments (see comments above in code)
    2. Tuning parameters: epsilon, L (Nstep), mass matrix
    3. Custom functions required: logDENS and glPOSTERIOR for log posterior and its gradient, respectively
96. Simple HMC example – perhaps the Gamma distribution with 2 parameters from my prelim
    * 1. (1p) Likelihood and priors. Log posterior
      2. (1p) Derive gradient for log posterior
      3. (1p) R code for logDENS and glPOSTERIOR.
      4. (1p) Leapfrog method
      5. (1p) Run example with set tuning parameters – don’t describe how to tune yet, just give the parameters and run
97. HMC practical considerations
    1. Tuning
       1. 1p: Why more challenging than MH. Compare number of parameters in MH to HMC (epsilon, L, mass matrix)

1p: introduction to step-by-step tuning  
Steps for tuning HMC:

1. Set the stepsize to an initial value such as ε = 1e − 2 with L = 10 and M unit diagonal

2. Run a preliminary HMC chain and compute the acceptance rate. Adjust ε until the acceptance is between 0.6 and 0.9.

3. Check the autocorrelation for each parameter either visually or via direct calculation. For parameters with high autocorrelation, reduce the relevant value in the diagonal of M. This adjustment may decrease the acceptance rate.

4. If necessary, increase L to further reduce autocorrelation in the simulation

1p: Automatic tuning via NUTS, mass matrix selection, epsilon heuristics. Focus on Stan software

* 1. Constrained parameters
     1. (1p) mathematical transformation to log scale (0, infty) => (-infty, infty)
     2. (1p) Neil’s constrained approach. Just focus on log.
     3. (1p) R code for leapfrog with constrained parameter
  2. QR decomposition (cite, tell why needed. Benefits, a little on how. Base function in R ‘qr’ )
     1. (1p) What is QR decomposition.
     2. (1p) Why we transform to orthogonal space. Mention why useful for Euclidean HMC
     3. (multi-p) Derivation of priors using this approach – empirical Bayes (see HMC paper draft)
     4. R code to perform QR decomposition and description. Use standard R functions

1. HMC use in statistical models (LM, GLM, GLMM)
   1. (1p) Describe requirements to implement these models using HMC. Need likelihood derivation (usually already completed). Need prior specification. Need to derive gradient of log posterior
   2. (1p) Requirements appear simple on the surface. However, different gradient needed for any change (e.g. in prior).
   3. (1p) When implementing manually, helpful to use R package like pracma to test that gradient is derived and coded correctly.
   4. (1p) Automated gradient algorithms are used for this. Calculates gradient exactly. Mention Stan autodiff. Some performance penalty but only compared to comparable code in C++
2. HMC: data example GLM (maybe GLMM) – maybe Agresti?
   1. Describe data
   2. Model specification and prior specification
   3. Derive likelihood and log posterior
   4. Derive gradient of log posterior
   5. Create design matrix in R
   6. Develop R code for log posterior and gradient (custom functions)
   7. Test gradient function using pracma and dummy code.
   8. Tune model using approach specified above
   9. Examine results. Compare to frequentist version
3. Discussion – highlight the contribution, why impt
   1. HMC expanding possibilities of complex statistical models. Flexibility of MH but improved efficiency
   2. Learning HMC using R implementation helpful to understand production environments such as Stan
   3. Next frontiers in HMC. Riemannian HMC, GPU
4. Introduction. (A new tool for Bayesian computation; running HMC in R – benefits and challenges; how this tutorial would help – emphasizing the educational aspects of running HMC in R)
5. The basic concepts and the generic algorithm (the MH and its limitations; the basic ideas of HMC; the algorithm; HMC properties; leapfrog, etc)
6. A general purpose HMC function in R (briefly describe the structure of this function)
7. Tuning of HMC in R (automatic tuning)
8. HMC in commonly used statistical models (succinctly describe its application in LM, GLM, and GLMM, stay away from semiparametric models)
9. Discussion.

Brief on MH: focus on the issues of MH (motivation for HMC). Especially large number of parameters – newer models. Carefully handled – not 3-4 pages.

Still have MH algorithm

Example: give impression usable to different types of models

Package on hmclearn…..here’s the code and download package

(separate)

First finish the package get done

Pick and choose into publishable papers

Simulation: linear, mixed effect, logistic model

* + - * Fit standard approach (MH)
      * Then fit mine (semipar) and contrast
      * Outcome:
        1. Faster convergence
        2. Similar results
        3. Use my own functions (hmclearn)