Part 5

Handling large datasets

ST740

North Carolina State University

What to do for massive datsets?

- MCMC remains the gold standard for Bayesian computing, but it can be slow
- However, many new tools have come online in the past ten years
- Bayesian computing for large datasets remains an active area of research

Outline

- **MAP estimation**
- Approximate likelihood
- Divide and Conquer
- Variational Bayes
- Stochastic gradient MCMC

MAP estimation

- Sometimes, especially for $n >> p$, quantifying parametric uncertainty is not important
- A MAP estimator is sufficient for point prediction
- The MAP estimator is

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \log(f(\mathbf{y}|\boldsymbol{\theta})) + \log(\pi(\boldsymbol{\theta}))$$

- Frequentists might call this a penalized likelihood where the prior is the penalty term
- The MAP estimator still incorporates prior information
- All optimization routines (EM, SGD, MM, etc) can be applied

Outline

- MAP estimation
- **Approximate likelihood**
- Divide and Conquer
- Variational Bayes
- Stochastic gradient MCMC

Approximate likelihood

- Approximations can be devised on a case-by-case basis
- For example, consider the geostatistical model with observation Y_i at spatial location s_i
- We might assume Y is a Gaussian process with mean $\mathrm{E}(Y_i) = \mu$, variance $\mathrm{V}(Y_i) = \sigma^2$ and correlation $\mathrm{Corr}(Y_i, Y_j) = \exp(-d_{ij}/\phi)$ for distance $d_{ij} = ||s_i - s_j||$
- The likelihood is $\mathbf{Y} = (Y_1, ..., Y_n) \sim \mathrm{Normal}(\mu, \sigma^2\Sigma)$ for $n \times n$ correlation matrix Σ with (i, j) element $\exp(-d_{ij}/\phi)$
- Dealing with the $n \times n$ covariance matrix is $O(n^3)$

Approximate likelihood

- The likelihood for $\boldsymbol{\theta} = (\mu, \sigma, \phi)$ can be written

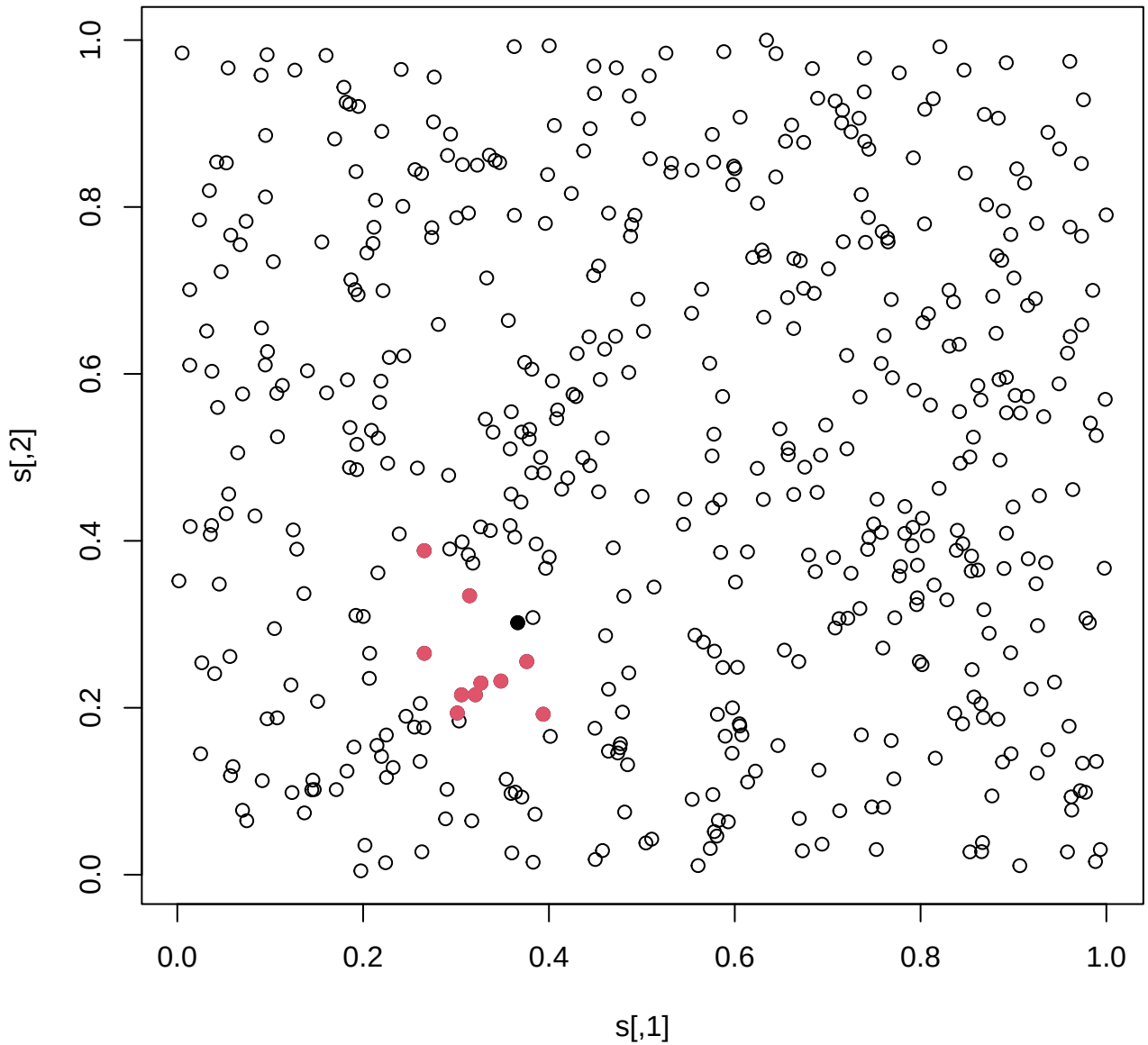
$$f(y_1, ..., y_n|\boldsymbol{\theta}) = \prod_{i=1}^{n} f(y_i|\boldsymbol{\theta}, y_1, ..., y_{i-1})$$

- The Vecchia approximation defines a neighbor set $\mathcal{N}_i \subset \{1, ..., i-1\}$ so that

$$f(y_1, ..., y_n|\boldsymbol{\theta}) \approx \tilde{f}(y_1, ..., y_n|\boldsymbol{\theta}) = \prod_{i=1}^{n} f(y_i|\boldsymbol{\theta}, y_j \text{ for } j \in \mathcal{N}_i)$$

- Note $\tilde{f}$ is a valid PDF, and requires only $O(nm^2)$ where m is the maximum size of $\mathcal{N}_i$

Vecchia approximation



Outline

- MAP estimation
- Approximate likelihood
- **Divide and Conquer**
- Variational Bayes
- Stochastic gradient MCMC

Divide and conquer

- Parallel computing is one obvious solution to the massive data problem
- MCMC is inherently sequential, but often some steps can be done in parallel, e.g., onerous likelihood computations
- Divide and conquer methods better utilize parallel computing
- The idea is to split the data intro groups, fit the model separately by group and then combine the results
- This is similar to a meta analysis where studies on the same topic are combined into a meta estimator

Divide and conquer

- Say the model is $Y_i|\theta \overset{indep}{\sim} f(y|\theta)$
- We split the data into B batches, with $\mathbf{Y}_{(1)}, ..., \mathbf{Y}_{(B)}$ so that $\mathbf{Y} = (\mathbf{Y}_{(1)}, ..., \mathbf{Y}_{(B)})$
- Each batch is analyzed separately, giving B posteriors of the form $p(\theta|\mathbf{Y}_b)$ for $b \in \{1, ..., B\}$
- These computations can be done in parallel using MCMC or Bayes CLT
- How to combine them to approximate the full posterior $p(\theta|\mathbf{Y})$?

Divide and conquer

- This is straightforward if the prior and posterior in each batch are approximately Gaussian
- The posterior can be written

$$p(\boldsymbol{\theta}|\mathbf{Y}) \propto f(\mathbf{Y}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}) = \prod_{b=1}^{B} \left[f(Y_{(b)}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})^{1/B} \right]$$

- If the prior is $\boldsymbol{\theta} \sim \text{Normal}(\mu, \Sigma)$, then the powered Gaussian prior $\pi(\boldsymbol{\theta})^{1/B}$ is $\boldsymbol{\theta} \sim \text{Normal}(\mu, B\Sigma)$
- Using this prior in each batch, denote the (maybe approximate) posterior in batch b as

$$\boldsymbol{\theta}|\mathbf{Y}_{(b)} \sim \text{Normal}(M_b, V_b)$$

Divide and conquer

- Combining terms gives

$$p(\boldsymbol{\theta}|\mathbf{Y}) \propto \prod_{b=1}^{B} \exp\left[-\frac{1}{2}(\boldsymbol{\theta} - M_b)^T V_b^{-1}(\boldsymbol{\theta} - M_b)\right]$$

- Multiplying terms and completing the square gives

$$\boldsymbol{\theta}|\mathbf{Y} \sim \text{Normal}(P_B^{-1} Q_B, P_B^{-1})$$

where $P_B = \sum_{b=1}^{B} V_b^{-1}$ and $Q_B = \sum_{b=1}^{B} V_b^{-1} M_b$

- There are extensions for non-Gaussian posteriors and dependent data, but these are generally hard problems

Divide and conquer

- This method can also be applied for streaming data
- Say Y_b is the data collected at time b
- At time b the posterior is

$$\theta|\mathbf{Y}_1, ..., \mathbf{Y}_b \sim \text{Normal}(P_b^{-1}Q_b, P_b^{-1})$$

  where $P_b = \sum_{t=1}^{b} V_t^{-1}$ and $Q_b = \sum_{t=1}^{b} V_t^{-1}M_t$

- To update the posterior at time $b+1$ you simply make the updates $P_{b+1} = P_b + V_{b+1}^{-1}$ and $Q_{b+1} = Q_b + V_{b+1}^{-1}M_{b+1}$
- You do not have to store Y_{b+1} after these updates

Sequential Monte Carlo (SMC)/Particle filtering

- SMC is used for non-Gaussian posteriors
- In can be used for streaming data
- It can also be used for a static analysis that passes through a large dataset sequentially
- SMC only touches each observation once, as opposed to MCMC that uses the whole dataset each iteration
- We will present the simplest version here, there is a rich literature on SMC[1]

[1] e.g., https://link.springer.com/book/10.1007/978-1-4757-3437-9

Sequential Monte Carlo (SMC)/Particle filtering

- As with MCMC, SMC using samples $\theta_1, ..., \theta_S$ to approximate the posterior
- We call these "particles"
- Rather than treating the particles as exchangeable as in MCMC, SMC gives them weights, $w_1, ..., w_S$
- We then approximation the posterior using weighted means, variances, etc

$$\mathrm{E}(\boldsymbol{\theta}|\mathbf{Y}) \approx \frac{\sum_{s=1}^{S} w_s \boldsymbol{\theta}_s}{\sum_{s=1}^{S} w_s}$$

- Particles with small weight are "filtered out"

How many particles to we need?

- The effective sample size is

$$ESS = \frac{(\sum_{s=1}^{S} w_s)^2}{\sum_{s=1}^{S} w_s^2}$$

- Best case: $w_s = w$ for all s and $ESS = n$

- Worst case: $w_1 = w > 0$ and $w_s = 0$ for all $s > 1$ and $ESS = 1$

- You need ESS in the hundreds

How to generate the particles?

- A simple approach is prior sampling, $\theta_s \overset{iid}{\sim} \pi(\boldsymbol{\theta})$
- This will only work well when the prior resembles the posterior
- Another possibility is to use MCMC from a subset of the data, e.g., the first batch
- This is slower, but gives larger *ESS* since the particle distribution is likely to be more similar to the posterior

How to weight the particles?

- Say $\theta_s = (\theta_{s1}, ..., \theta_{sp}) \overset{iid}{\sim} \pi(\boldsymbol{\theta})$
- Define the weight after batch b as the likelihood

$$w_{bs} = \prod_{t=1}^{b} f(\mathbf{Y}_t|\boldsymbol{\theta}_s)$$

- The weights can be updated as the data arrive as

$$w_{b+1,s} = w_{b,s} f(\mathbf{Y}_b|\boldsymbol{\theta}_s)$$

- Only the weights and not the data need be stored
- Posterior summaries are fast to compute, e.g.,

$$\text{Prob}(\theta_j > 0|\mathbf{Y}_1, ..., \mathbf{Y}_b) \approx \frac{\sum_{s=1}^{S} w_{bs} I(\theta_{sj} > 0)}{\sum_{s=1}^{S} w_{bs}}$$

How to weight the particles?

- Say are MCMC samples from the posterior given the first batch

$$\theta_1, ..., \theta_S \sim p(\boldsymbol{\theta}|\mathbf{Y}_1)$$

- Define the weight after batch $b > 1$ as the likelihood

$$w_{bs} = \prod_{t=2}^{b} f(\mathbf{Y}_t|\boldsymbol{\theta}_s)$$

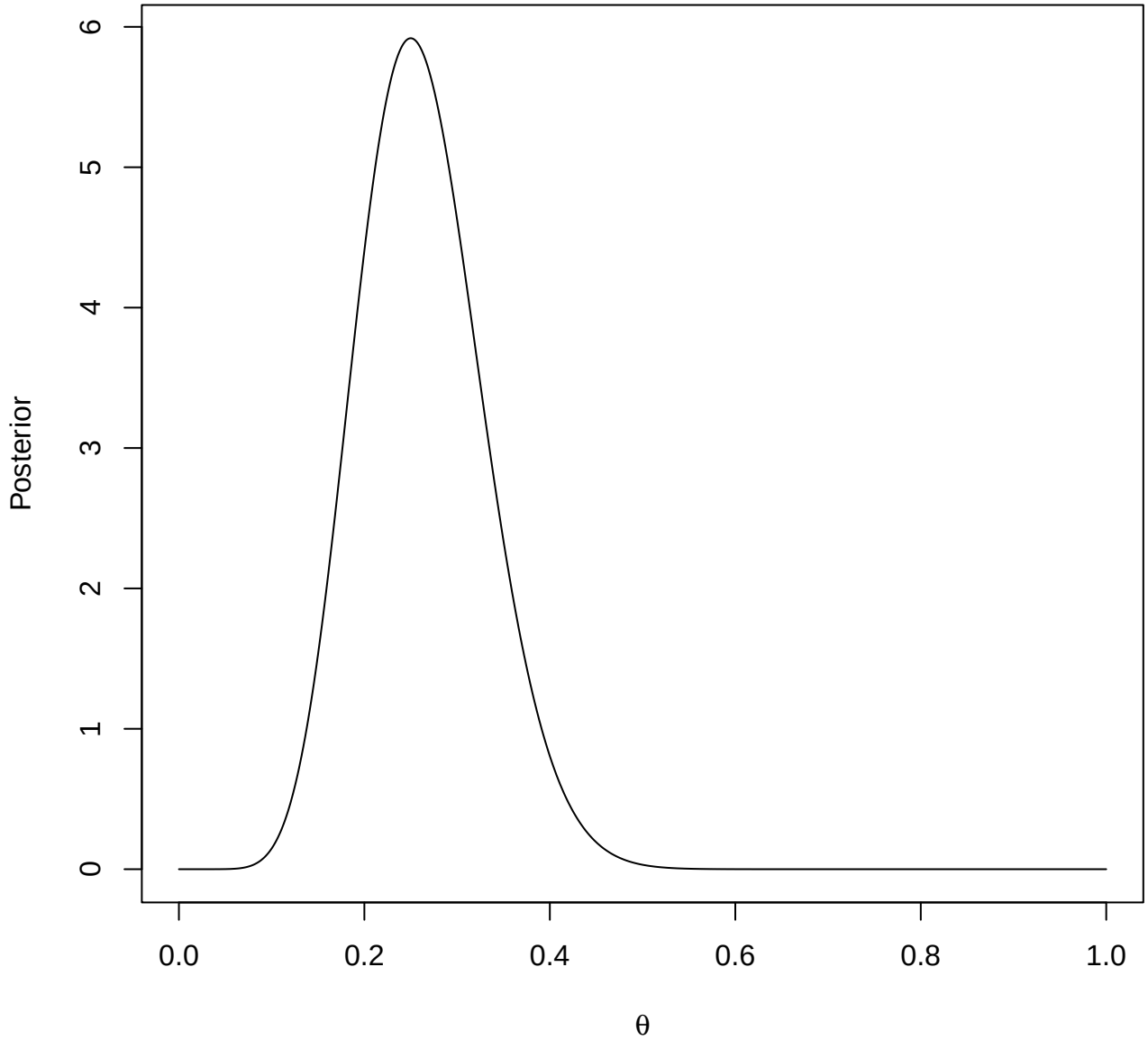
- The weights can be updated as the data arrive as

$$w_{b+1,s} = w_{b,s} f(\mathbf{Y}_b|\boldsymbol{\theta}_s)$$

SMC versus MCMC

```
> n <- 40
> Y <- 10
> a <- 1
> b <- 1
> t <- seq(0,1,length=1000)
> p <- dbeta(t,Y+a,n-Y+b)
> plot(t,p,type="l",xlab=expression(theta),
>      ylab="Posterior")
>
> (Y+a)/(n+a+b) # Exact E(theta|Y)
[1] 0.2619048
```

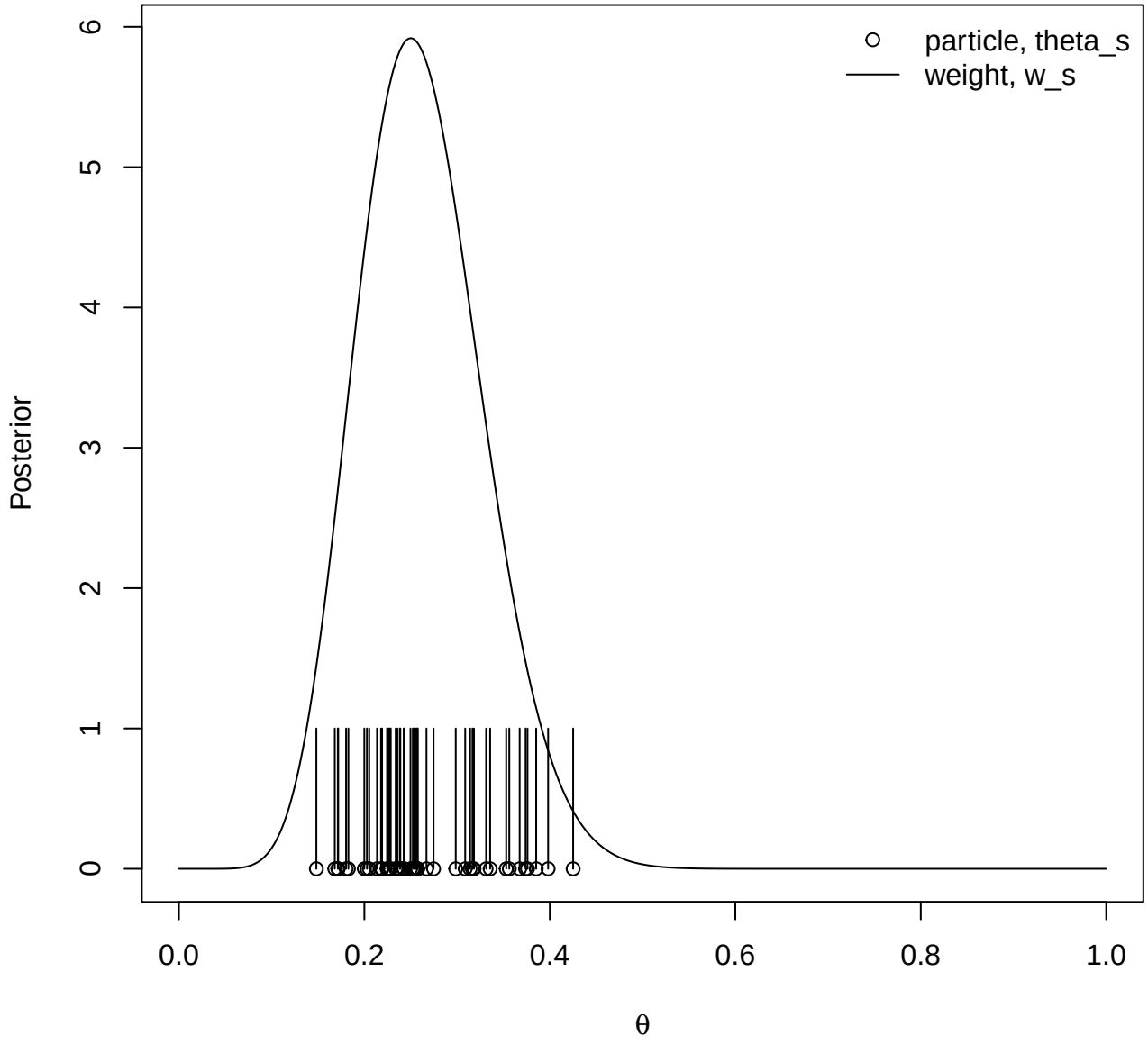
SMC versus MCMC



SMC versus MCMC

```
> # MCMC
> set.seed(919)
> S      <- 50
> theta <- rbeta(S,Y+a,n-Y+b)
> w      <- rep(1,S)
>
> plot(t,p,type="l",xlab=expression(theta),
>      ylab="Posterior")
> points(theta,rep(0,S))
> lines(theta,w,type="h")
> legend("topright",c("particle, theta_s",
>                     "weight, w_s"),
>        pch=c(1,NA),lty=c(NA,1),bty="n")
>
> mean(theta) # Approximate E(theta|Y)
[1] 0.2634136
```

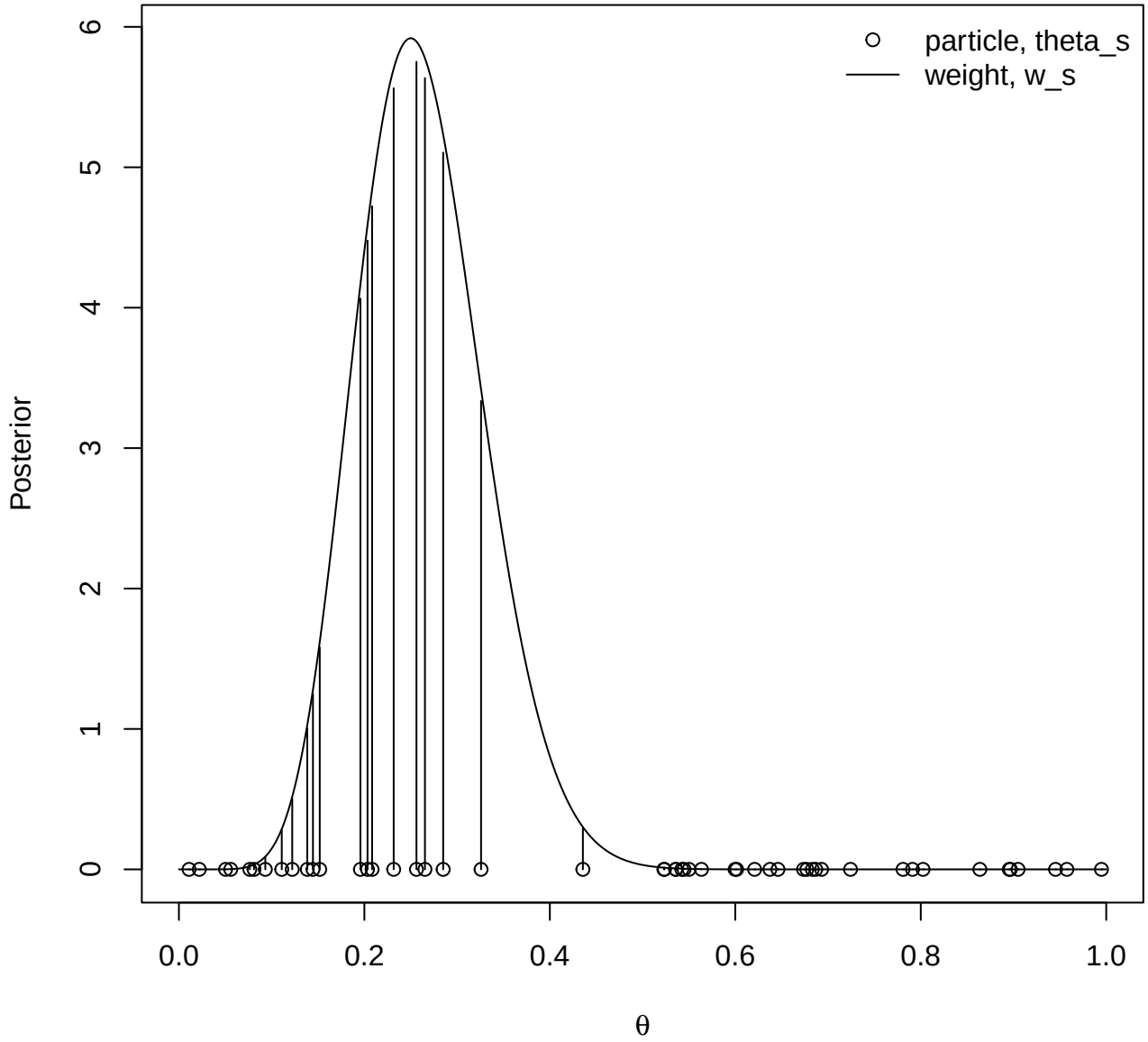
SMC versus MCMC



SMC versus MCMC

```
> # SMC
> set.seed(919)
> S      <- 50
> # Sample from prior
> theta <- rbeta(S,a,b)
> # Weight by likelihood
> w      <- 40*dbinom(Y,n,theta)
>
> plot(t,p,type="l",xlab=expression(theta),
>      ylab="Posterior")
> points(theta,rep(0,S))
> lines(theta,w,type="h")
> legend("topright",c("particle, theta_s",
>                     "weight, w_s"),
>        pch=c(1,NA),lty=c(NA,1),bty="n")
> sum(w*theta)/sum(w) # Approx E(theta|Y)
[1] 0.2355692
```

SMC versus MCMC



How to weight the particles?

- The particles are from the prior and weighted by the likelihood, so the weighted particles approximate the posterior
- However, the approximation is poor if *ESS* is small and only a few particles have most of the weight
- There are methods to replace low-weight particles with particles with more support

Outline

- MAP estimation
- Approximate likelihood
- Divide and Conquer
- **Variational Bayes**
- Stochastic gradient MCMC

Variational Bayes (VB)

- VB is popular in the machine learning community
- The main idea is the assume the posterior resides in a simple class of distributions, and then find the best approximation to the full posterior in this class
- For example, we might assume that

$$p(\boldsymbol{\theta}|\mathbf{Y}) \approx \prod_{j=1}^{p} q(\theta_j|\mathbf{v}_j)$$

  where q is the normal PDF with parameters $\mathbf{v}_j = (\mu_j, \sigma_j^2)$
- All that is left is to solve for the $\mathbf{v}_j$
- This is reminiscent of the Bayesian CLT

Variational Bayes

- Let $q(\boldsymbol{\theta}|\mathbf{v})$ be the approximate posterior
- The variational parameters are $\mathbf{v}$
- The most common assumption is the mean-field posterior

$$q(\boldsymbol{\theta}|\mathbf{v}) = \prod_{j=1}^{p} q_j(\theta_j|\mathbf{v}_j)$$

  that assumes posterior independence between the parameters

- Other approximations are possible, e.g., q could be multivariate normal with $\mathbf{v} = \{\mu, \Sigma\}$

Variational Bayes

- The variational parameters are selected to minimize the KL divergence between p and q

$$KL(q||p) = \int \log\left[\frac{q(\boldsymbol{\theta}|\mathbf{v})}{p(\boldsymbol{\theta}|\mathbf{Y})}\right] q(\boldsymbol{\theta}|\mathbf{v}) d\boldsymbol{\theta}$$

- Writing $p(\boldsymbol{\theta}|\mathbf{Y}) = f(\mathbf{Y}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})/m(\mathbf{y})$, $KL(q||p)$ is

$$\int \log\left[\frac{q(\boldsymbol{\theta}|\mathbf{v})}{f(\mathbf{Y}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}))}\right] q(\boldsymbol{\theta}|\mathbf{v}) d\boldsymbol{\theta} + \log(m(\mathbf{Y}))$$

- The integral term is the evidence lower bound (ELBO)
- It can be shown that ELBO $\leq \log(m(\mathbf{Y}))$
- The term $\log(m(\mathbf{Y}))$ can be ignored for estimating $\mathbf{v}$ and we minimize the ELBO

Variational Bayes

- The goal is to find the values of $\mathbf{v}$ to minimize

$$ELBO(\mathbf{v}, \mathbf{Y}) = \int \log \left[\frac{q(\boldsymbol{\theta}|\mathbf{v})}{f(\mathbf{Y}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}))} \right] q(\boldsymbol{\theta}|\mathbf{v}) d\boldsymbol{\theta}$$

- Sometimes the solution for $\mathbf{v}$ has a closed form
- More often you use coordinate descent, where you optimize the elements of $\mathbf{v}$ one at a time with the others held fixed at their current value
- This is similar to MCMC, except each iteration is an optimization rather than a sample
- Sometimes these univariate updates have a closed-form, sometimes they require numerical optimization

Variational Bayes

- VB is generally orders of magnitude faster than MCMC
- Unlike MAP estimation, VB gives a posterior variance
- However, it often underestimates the variance because of simplifying assuming such as normality and independence
- This may not be a concern for massive datasets
- Often this can be resolved by finding a reparameterization of the parameters so that the simplifying assumptions hold

Stochastic gradient descent (SGD)

- MAP and VB estimation often use SGD for large datasets
- The MAP estimator for **independent** data has the form

$$\begin{aligned}\hat{\theta}_{MAP} &= \underset{\theta}{\text{argmin}} \sum_{i=1}^{n} -[\log\{f(Y_i|\theta) + \log\{\pi(\theta)\}/n] \\ &= \underset{\theta}{\text{argmin}} \sum_{i=1}^{n} l(\theta|Y_i)\end{aligned}$$

- Similarly, the EB estimator can be written

$$\hat{\mathbf{v}} = \underset{\mathbf{v}}{\text{argmin}} \sum_{i=1}^{n} Q(\mathbf{v}|Y_i)$$

- For large n, computing this sum is slow and SGD is faster

Gradient descent (GD)

- GD is a classic optimization method
- Let $\nabla_i(\boldsymbol{\theta})$ be the gradient vector of $l(\theta|\, Y_i)$ with j^{th} element

$$\frac{\partial}{\partial \theta_j} l(\boldsymbol{\theta}|\, Y_i)$$

- The full gradient is $\nabla(\boldsymbol{\theta}) = \sum_{i=1}^{n} \nabla_i(\boldsymbol{\theta}_t)$
- GD begins with initial value $\boldsymbol{\theta}_0$ and updates $\boldsymbol{\theta}$ as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \nabla(\boldsymbol{\theta}_t)$$

- The step-size/learning rate η is a tuning parameter

Stochastic gradient descent (SGD)

- SGD uses random subset of observations to approximate the gradient
- Let $\mathcal{A} \subset \{1, ..., n\}$ be a random subset of indices with m elements
- An unbiased estimator of the gradient is

$$\frac{n}{m} \sum_{i \in \mathcal{A}}^{n} \nabla_i(\boldsymbol{\theta}_t) = \sum_{i=1}^{n} \nabla_i(\boldsymbol{\theta}_t)$$

- SGD averages over minibatches of data

$$\{Y_i; i \in \mathcal{A}\}$$

Stochastic gradient descent (SGD)

SGD begins with initial value θ_0 and executes E epochs/iterations with the following steps for epoch t

0 Randomly partition $\{1, ..., n\}$ to minibatches $\mathcal{A}_1$,...,$\mathcal{A}_B$

1 Set $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \eta_t \frac{n}{|\mathcal{A}_1|} \sum_{i \in \mathcal{A}_1} \nabla_i(\boldsymbol{\theta}_{t-1})$

2 Set $\boldsymbol{\theta}_t = \boldsymbol{\theta}_t + \eta_t \frac{n}{|\mathcal{A}_2|} \sum_{i \in \mathcal{A}_2} \nabla_i(\boldsymbol{\theta}_t)$

3 ...

B Set $\boldsymbol{\theta}_t = \boldsymbol{\theta}_t + \eta_t \frac{n}{|\mathcal{A}_B|} \sum_{i \in \mathcal{A}_B} \nabla_i(\boldsymbol{\theta}_t)$

Stochastic gradient descent (SGD)

- The main tuning parameters are the minibatch size and the learning rate, η_t
- A common minibatch size is often 32 (so $B \approx n/32$)
- Usually η_t decreases with t to balance exploration and refinement
- It must[2] satisfy $\sum_{t=1}^{\infty} \eta_t = \infty$ and $\sum_{t=1}^{\infty} \eta_t^2 < \infty$
- A common learning rate schedule is $\eta_t = c/(1+t)$ for tuning parameter c

[2] Robbins-Monro

SGD extensions

- **Line search** sets

$$\eta_t = \arg\min_{\eta} \sum_{i \in \mathcal{A}_b} l\{\boldsymbol{\theta}_t + \eta \nabla(\boldsymbol{\theta}_t) | Y_i\}$$

- **AdaGrad** adapts the learning rate for individual parameters

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \eta_t \text{Diag}\left[\text{Diag}\left\{\tilde{H}(\boldsymbol{\theta})\right\}\right]^{-1/2} \nabla(\theta)$$

where $\tilde{H}(\boldsymbol{\theta}) = \nabla(\theta)^T \nabla(\theta)$ approximates the Hessian

SGD extensions

- **Momentum** sets

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta_t \nabla(\theta_t) + \alpha_t(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1})$$

  where tuning parameter α_t controls momentum

- **Dropout** randomly sets some elements of $\boldsymbol{\theta}_t$ to zero at each step

- **Adaptive Moment Estimation (Adam)** combines many of these ideas and is the most common approach

Outline

- MAP estimation
- Approximate likelihood
- Divide and Conquer
- Variational Bayes
- **Stochastic gradient MCMC**

Stochastic gradient MCMC (SGMCMC)

- MALA and HMC use the posterior's gradient to generate high-quality candidates for MH sampling
- Computing the gradients is slow when n is large
- SGMCMC approximates the gradient with random subsamples of the data
- Rather than use the full dataset for the acceptance probability in an MH step, SGMCMC uses a small step size and accepts all steps
- You can also use SGMCMC to generate candidates and the full data for the MH step

Stochastic gradient MCMC

- We will discuss Langevin dynamics, but the ideas apply to HMC as well
- The posterior is $p(\boldsymbol{\theta}|\mathbf{Y}) \propto \left[\prod_{i=1}^{n} f(Y_i|\boldsymbol{\theta})\right] \pi(\boldsymbol{\theta})$
- Write this as

$$p(\boldsymbol{\theta}|\mathbf{Y}) \propto \exp[-U(\boldsymbol{\theta})] = \exp\left[\sum_{i=1}^{n} U_i(\boldsymbol{\theta})\right]$$

  where $U_i(\boldsymbol{\theta}) = \log[f(Y_i|\boldsymbol{\theta})] - \log[\pi(\boldsymbol{\theta})]/n$

- $U(\boldsymbol{\theta})$ is called the potential function

Stochastic gradient MCMC

- Langevin diffusion satisfies the SDE

$$d\theta(t) = -\frac{1}{2}\nabla U[\theta(t)] + dB(t)$$

where $B(t)$ is Brownian motion

- For initial value $\theta(0)$, step size h and $Z(t) \sim \text{Normal}(\mathbf{0}, \mathbf{I}_p)$

$$\theta(t+h) = \theta(t) - \frac{h}{2}\nabla U[\theta(t)] + \sqrt{h}Z(t)$$

gives samples with stationary distribution $\approx p(\theta|\mathbf{Y})$

Stochastic gradient MCMC

- The gradient term is

$$\nabla U[\theta(t)] = \sum_{i=1}^{n} \nabla U_i[\theta(t)]$$

- SGMCMC uses a random subset of observations to approximate the gradient
- Say $\mathcal{N}(t) \subset \{1, ..., n\}$ with m labels selected at random without replacement for each t
- The gradient term is approximated as

$$\nabla U[\theta(t)] \approx \frac{n}{m} \sum_{i \in \mathcal{N}(t)} \nabla U_i[\theta(t)]$$

- Samples approximately follow $p(\theta|\mathbf{Y})$ for small h[3]

[3] https://arxiv.org/abs/1907.06986