Particle Swarm Optimization Assisted Metropolis Hastings Algorithms

Abstract

Fitting dependent data models is often a challenging endeavor and often requires some form of dimension reduction or customized estimation algorithm. Particle swarm optimization (PSO) refers to a class of heuristic optimization algorithms that exploit analogies with animal flocking behavior in order to obtain optima without strong conditions on the objective function. We introduce two new classes of PSO algorithms, termed adaptively tuned PSO (AT-PSO) and adaptively tuned bare bones PSO (AT-BBPSO). In both algorithms we add a dynamically tuned parameter to previously existing PSO algorithms. We propose using these PSO algorithms to approximate Bayesian posterior distributions in order to construct efficient proposals for independent Metropolis-Hastings and independent Metropolis-Hastings within Gibbs algorithms. For the latter, we propose using a global approximation to the posterior in order to construct an approximation to the conditional distribution which requires a Metropolis step, thereby requiring the optimization algorithm only once rather than every iteration of the Markov chain Monte Carlo (MCMC) algorithm. In order to illustrate our method and compare it to alternatives, we provide a simulation study and apply it to constructing MCMC algorithms to estimate reduced rank spatial models of American Community Survey (ACS) 5-year estimates of county populations in the United States.

KEY WORDS: Bayesian estimation; Markov chain Monte Carlo; Official statistics; Optimization; Spatial data

1 Introduction

[REWRITE THIS PARAGRAPH - IS TALL DATA WORTH MENTIONING?] One common type of big data problem may be called "tall" data — that is many observations of a relatively small number of variables. Bayesian estimation of tall data models can be particularly challenging in the dependent data and non-Gaussian settings due to the need to estimate or integrate out a high dimensional latent process, e.g., with dimension equal to the size of the dataset. Sometimes the latent process can be written as a function of a relatively small number of latent random variables, but, even in this case, standard Markov chain Monte Carlo (MCMC) techniques to approximate the posterior are often still slow. A common alternative is to leverage normal approximations to the posterior to obtain approximations to the marginal distributions of each parameter, e.g., INLA (Rue et al., 2009). This is approximation is often very good and much faster than MCMC, though neither advantage is universally true, particularly when features of the joint posterior distribution rather than just the marginals are desired or when the parameter space is too large (e.g., see Taylor and Diggle (2014)). [CHECK ON THE JOINT VS MARGINAL THING AND OTHER INLA LIMITATIONS — CITATIONS!!!] Another common strategy is to use Hamiltonian Monte Carlo (HMC) (Neal et al., 2011), particularly the No-U-Turn sampler (NUTS) (Homan and Gelman, 2014) which is implemented in the Stan software (Carpenter et al., 2015). However HMC requires many log posterior evaluations per iteration and in the tall data setting these are typically expensive.

We propose using old MCMC technology improved by new optimization techniques. It is well known that a Bayesian posterior distributions for a fixed set of parameters tends asymptotically to a normal distribution centered at the posterior mode as the sample size increases; see Schervish (1997, Chapter 7.4). This normal approximation, often called a Laplace approximation, is used by INLA but is also frequently used to construct independent

Metropolis-Hastings (IMH) samplers (Metropolis et al., 1953; Hastings, 1970) — typically using a t proposal instead of a normal. IMH samplers based on a good approximation to the posterior distribution typically have high acceptance rates along with fast mixing and convergence, but it is often impractical to find an adequate approximation. Even in cases where the normal approximation is appropriate the posterior may be too high dimensional for this approach to be practical. In larger models numerical methods are usually required to find the posterior mode and sometimes also to integrate out a latent process, but standard methods for doing so are usually prohibitively slow or fail outright. However, new heuristic optimization algorithms tend to work reasonably well over a wider class of objective functions and in larger parameter spaces, for example, genetic algorithms (Goldberg and Holland, 1988) and particle swarm optimization (PSO) (Clerc and Kennedy, 2002; Blum and Li, 2008; Clerc, 2010). We propose using PSO to obtain the posterior mode so that effective independent Metropolis samplers can be constructed in a wider range of models. Other heuristic optimization algorithms may be useful for the same task, but we do not explore them here. We also develop several novel PSO algorithms that help obtain good estimates of the posterior mode and that may also have utility in other optimization contexts. We illustrate our method on two different modeling applications. The first application considers a group of reduced rank spatial models of American Community Survey (ACS) 5-year period estimates of county population estimates for the U.S. in 2014. In the second application, we adapt a hierarchical model for predicting the outcome of the 1988 presidential election from Gelman and Hill (2006). We fit these models using our proposed independent Metropolis-Hastings algorithm, an independent Metropolis within Gibbs algorithm based on the same Laplace approximation, and a variety of other MCMC algorithms. We compare the computational cost for each of the algorithms and discuss their ease of use.

[REWRITE THIS PARAGRAPH - SECTIONS HAVE CHANGED] The remainder of this paper is organized as follows. Section 2 introduces PSO along with our novel PSO

algorithms and reports the results of testings these algorithms on a suite of standard test functions. Section 3 describes the IMH algorithms we construct with the aid of PSO whereas Section 4 describes a general strategy for using our method to fit generalized linear mixed models and details our applications. Section 6 compares our MCMC techniques to a variety of other techniques for our applications, and Section 7 concludes with discussion.

2 Particle swarm optimization

We briefly describe PSO here; refer to Blum and Li (2008) for an excellent introduction and Clerc (2010) for more detail. Suppose that we wish to optimize some objective function $Q(\boldsymbol{\theta}) : \Re^D \to \Re$ — without loss of generality we will assume the goal is maximization. Let $i = 1, 2, \ldots, n$ index a set of particles over time, $t = 1, 2, \ldots, T$, where each particle consists of a location at every period $\boldsymbol{\theta}_i(t) \in \Re^D$, a velocity $\boldsymbol{v}_i(t) \in \Re^D$, a personal best location $\boldsymbol{p}_i(t) \in \Re^D$, and a group best location $\boldsymbol{g}_i(t) \in \Re^D$. Here we mean "best" in the sense of maximizing Q, so the objective function evaluated at $\boldsymbol{p}_i(t)$ is larger than at any point in particle i's history. More formally $Q(\boldsymbol{p}_i(t)) \geq Q(\boldsymbol{\theta}_i(s))$ for any $s \leq t$. The group best location is defined with respect to some neighborhood $\mathcal{N}_i$ of particle i; that is, $\boldsymbol{g}_i(t) = \arg\max_{\{\boldsymbol{p}_j | j \in \mathcal{N}_i\}} Q(\boldsymbol{p}_j(t))$. In the simplest case where the entire swarm is the neighborhood of each particle, $\boldsymbol{g}_i(t) \equiv \boldsymbol{g}(t) = \arg\max_{j \in 1:n} Q(\boldsymbol{p}_j(t))$. The generic PSO algorithm updates as follows:

$$
\begin{aligned}
\boldsymbol{v}_i(t+1) &= \omega \boldsymbol{v}_i(t) + \phi_1 \boldsymbol{r}_{1i}(t) \circ \{\boldsymbol{p}_i(t) - \boldsymbol{\theta}_i(t)\} + \phi_2 \boldsymbol{r}_{2i}(t) \circ \{\boldsymbol{g}_i(t) - \boldsymbol{\theta}_i(t)\}, \\
\boldsymbol{\theta}_i(t+1) &= \boldsymbol{\theta}_i(t) + \boldsymbol{v}_i(t+1), \\
\boldsymbol{p}_i(t+1) &= \begin{cases} \boldsymbol{p}_i(t) & \text{if } \; Q(\boldsymbol{p}_i(t)) \geq Q(\boldsymbol{\theta}_i(t+1)) \\ \boldsymbol{\theta}_i(t+1) & \text{otherwise,} \end{cases} \\
\boldsymbol{g}_i(t+1) &= \arg \max_{\{\boldsymbol{p}_j(t+1) | j \in \mathcal{N}_i\}} Q(\boldsymbol{p}_j(t+1)),
\end{aligned} \tag{1}
$$

where $\circ$ denotes the Hadamard product (element-wise product), $\boldsymbol{r}_{1i}(t)$ and $\boldsymbol{r}_{2i}(t)$ are each vectors of D random variates independently generated from the $U(0,1)$ distribution, and $\omega > 0$, $\phi_1 > 0$, and $\phi_2 > 0$ are user-defined parameters. The term $\omega \boldsymbol{v}_i(t)$ controls the particle's tendency to keep moving in the direction it is already going, so ω is called the inertia parameter. For $\omega < 1$ velocities tend to decrease over time, while for $\omega > 1$ they tend to increase over time. Similarly $\phi_1 \boldsymbol{r}_{1i}(t) \circ (\boldsymbol{p}_i(t) - \boldsymbol{\theta}_i(t))$ controls the particle's tendency to move towards its personal best location while $\phi_2 \boldsymbol{r}_{2i}(t) \circ (\boldsymbol{g}_i(t) - \boldsymbol{\theta}_i(t))$ controls its tendency to move toward the group's best location, so ϕ_1 and ϕ_2 are called the cognitive correction factor and social correction factor, respectively (Blum and Li, 2008). This version of PSO is equivalent to Clerc and Kennedy (2002)'s constriction type I particle swarm. There are many variants of the PSO algorithm, often obtained through choosing (ω, ϕ_1, ϕ_2) in special ways or sometimes even dynamically. A default version of the algorithm sets $\omega = 0.7298$ and $\phi_1 = \phi_2 = 1.496$; see Clerc and Kennedy (2002) and Blum and Li (2008) for justification of these choices. Even when $\omega < 1$, if ϕ_1 and ϕ_2 are set high enough the velocities of the particles can continually increase and cause the swarm to make jumps that are much too large. A heavy handed way to solve this problem is by setting an upper bound on the velocity of any particle in any given direction, called velocity clamping. However, the default parameter values suggested by Clerc and Kennedy (2002) are also designed to prevent exactly this sort of velocity explosion.

Any PSO variant can also be combined with various neighborhood topologies that control how the particles communicate to each other. The default global topology allows each particle to see each other particle's previous best location for the social components of their respective velocity updates, but this can cause inadequate exploration and premature convergence. Alternative neighborhood topologies limit how many other particles each particle can communicate with. For example, particle 5 may only look at itself and particles 4 and 6 when determining what its group best location is. This allows information about high value

locations in the domain of the objective function to eventually reach every particle in the swarm, but much more slowly so that each particle has an opportunity to explore the space more fully first. Appendix Appendix A: contains a short description of the ring topologies, but there are many alternatives in the literature.[CITATION]

A variant of PSO, the bare bones PSO algorithm (BBPSO) is a PSO algorithm introduced by Kennedy (2003) that strips away the velocity term for simplification and removes the need for the user to choose parameters outside of the swarm size. Let $\theta_{ij}(t)$ denote the jth coordinate of the position for the ith particle in period t, and similarly for $p_{ij}(t)$ and $g_{ij}(t)$. Then the BBPSO algorithm obtains a new position coordinate θ_{ij} via

$$\theta_{ij}(t+1) \sim N\left(\frac{p_{ij}(t)+g_{ij}(t)}{2}, |p_{ij}(t)-g_{ij}(t)|^2\right) \tag{2}$$

where $N(\mu, \sigma^2)$ is the normal distribution with mean μ and standard deviation σ^2. The updates of $\boldsymbol{p}_i(t)$ and $\boldsymbol{g}_i(t)$ are the same as in (1). There are several variants of this algorithm proposed including using distributions from different location-scale families — e.g., using the t-distribution and modifying the location or the scale parameters (for example Krohling et al. (2009), Hsieh and Lee (2010), Richer and Blackwell (2006), and Campos et al. (2014)). A commonly used variant of BBPSO also introduced by Kennedy (2003), sometimes called BBPSOxp, gives each coordinate of $\boldsymbol{\theta}_i(t+1)$ a 0.5 probability of moving to its group best location on that coordinate; i.e.,

$$\theta_{ij}(t+1) = \begin{cases} N\left(\frac{p_{ij}(t)+g_{ij}(t)}{2}, |p_{ij}(t)-g_{ij}(t)|^2\right) & \text{with probability 0.5} \\ g_{ij}(t) & \text{otherwise.} \end{cases}$$

Appendix Appendix A: contains more detail on the BBPSO variants in the literature which we employ.

2.1 Adaptively tuned BBPSO

BBPSO adapts the size effective search space of the swarm over time through the variance term: $|p_{ij}(t)-g_{ij}(t)|^2$. As the personal best locations of the swarm move closer together, these variances decrease and the swarm tries locations which are closer to known high value areas in the space. This behavior is desireable, but the adaptation is forced to occur through one channel: the personal and group best locations. If the personal best locations of the swarm are arranged in a rough ring around the global optimum, smaller variances are desireable so that the new particle locations have a tendency to be in the center of the ring. On the other hand, if the personal best locations of the swarm are all to one side of the optimum and fairly far away, larger variances are desireable so that the particles can quickly approach the neighborhood of the optimum. BBPSO cannot distinguish between these two cases.

In order to allow it to adapt in a more flexible manner, we modify the BBPSO variance to $\sigma^2(t)|p_{ij}(t) - g_{ij}(t)|^2$ and tune $\sigma(t)$ in a manner similar to adaptive random walk Metropolis algorithms (Andrieu and Thoms, 2008). Define the improvement rate of the swarm in period t as $R(t) = \#[\{i : Q(\boldsymbol{p}_i(t)) > Q(\boldsymbol{p}_i(t-1))\}]/n$ and let R^* denote the target improvement rate. Then what we term adaptively tuned BBPSO (AT-BBPSO) updates as follows:

$$\theta_{ij}(t+1) \sim t_{df}\left(\frac{p_{ij}(t)+g_{ij}(t)}{2}, \sigma^2(t)|p_{ij}(t)-g_{ij}(t)|^2\right) \quad (3)$$

$$\log\sigma^2(t+1) = \log\sigma^2(t) + c\{R(t) - R^*\}$$

where *df* is a user chosen degrees of freedom parameter. We use a t kernel instead of a Gaussian in order to allow for more flexibility. Both using a t kernel and adding a fixed scale parameter have been discussed in the BBPSO literature, but as Kennedy (2003) notes, something about the normal with $\sigma = 1$ is special that causes the algorithm to work well. When $\sigma(t)$ is dynamically adjusted, however, we find the additional flexibility useful.

Another similar PSO algorithm in the literature is Hsieh and Lee (2010) which proposes

a modified version of BBPSO with

$$\theta_{ij}(t+1) \sim N\left(\omega_1 \frac{p_{ij}(t)+g_{ij}(t)}{2}, \omega_2 |p_{ij}(t)-g_{ij}(t)|^2\right),$$

where $\omega_1 \leq 1$ and $\omega_2 \leq 1$ are constriction parameters that are eventually both set to one after enough iterations of the algorithm. The authors suggest dynamically adjusting the constriction parameters in the early stage of the algorithm before they are set to one, but give no suggestion for how to do this.

We propose a variant of this algorithm where $\omega_1 = 1$ and where $\omega_2 \equiv \sigma$ is a dynamically adjusted scale parameter, and additionally we propose using a more general student's t-distribution. Given a value for $\sigma(t)$ in period t, $\theta_{ij}(t+1)$ is obtained via

$$\theta_{ij}(t+1) \sim t_{df}\left(\frac{p_{ij}(t)+g_{ij}(t)}{2}, \sigma(t)|p_{ij}(t)-g_{ij}(t)|^2\right) \tag{4}$$

where df is a degrees of freedom parameter. Define the improvement rate of the swarm in period t as $R(t) = \#[\{i : Q(\boldsymbol{p}_i(t)) > Q(\boldsymbol{p}_i(t-1))\}]/n$, where if A is a set then $\#(A)$ is the number of members of that set. Heuristically, we can think about this rate similar to a random walk Metropolis acceptance rate. If the rate is too large, the jumps we are proposing are too small; so, while each particle is likely to find a new best location, the improvement will tend to be small. Similarly, if the rate is too small the jumps are too large and very few improvements occur.

The rates we observe will depend on particular parameter values we choose for the algorithm, but using the above intuition we can automatically and adaptively tune the scale parameter. Let $\lambda(t) = \log \sigma(t)$ and R^* denote the target improvement rate. Then we update $\lambda(t)$ to $\lambda(t+1)$ each iteration via $\lambda(t+1) = \lambda(t) + c \times sgn(R(t) - R^*)$ where

$$sgn(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

and c is some user-defined constant, e.g., $c = 0.1$. Call BBPSO algorithms which use this adaptively tuned BBPSO (AT-BBPSO — e.g., AT-BBPSO-MC and AT-BBPSOxp-MC). One nice feature of this algorithm is that it tunes the effective search space adaptively as the algorithm gets closer to the optimum. To some degree this already occurs with the $|p_{ij} - g_{ij}|$ term in the standard deviation. However, once BBPSO has approximately converged a more targeted local search may still yield improvement. AT-BBPSO increases the number of user-defined parameters relative to BBPSO, but they are fairly easy to select. These parameters are $\lambda(0)$, c, swarm size, R^*, and the degrees of freedom of the t-distribution. The value of $\lambda(0)$ has little effect on the algorithm since it adapts fairly quickly, so the user can safely leave it at a default value. We set $\lambda(0) = 0$ since that initializes the AT-BBPSO algorithm at an equivalent BBPSO algorithm. The value of c has a slightly higher impact, but again it is relatively small. It controls how fast $\lambda(t)$ adapts and how precisely it can adapt. A larger value of c causes $\lambda(t)$ to adapt more quickly, but it also means that the set of possible values that $\lambda(t)$ can take on is smaller. Since $\lambda(t)$ can only take on values on the the lattice $\{\ldots, \lambda(0) - 2c, \lambda(0) - c, \lambda(0), \lambda(0) + c, \lambda(0) + 2c, \ldots\}$, a larger value of c makes it more likely that $\lambda(t)$ cannot get close to the value which yields the target improvement rate, which can cause the swarm to move toward the maximum more slowly. Optimizing this parameter can improve the algorithm, but in practice we find that the gains are small compared to just setting $c = 0.1$.

The swarm size is a more difficult parameter to choose, though all PSO algorithms share it in common. The basic tradeoff is the classic accuracy versus speed tradeoff. A larger swarm size yields better estimates of the maximum but at the cost of more function evaluations. To some extent increasing the swarm size is a substitute for running the algorithm for more iterations, though the number of iterations should typically be at least an order of magnitude larger than the swarm size. The consequential parameters of the AT-BBPSO algorithms are R^* and df. The value of R^* should be fairly small so that the algorithm can

more easily jump between local maxima, yet large enough that it can converge on the global maximum once in its region. In our experience setting R^* from 0.3 to 0.5 tends to yield algorithms which do the best job of finding the global max, but since $R(t)$ can only take on values in a discrete set R^* should also be set in concert with the size of the swarm. For example, if there are only 100 members of the swarm, then $R(t)$ can only take on the values $0/100, 1/100, 2/100, \ldots, 100/100$ so setting $R^* = 0.015$ or $R^* = 0.019$ will result in the same behavior. The degrees of freedom parameter also should typically be set fairly small, e.g., $df = 1$. Tinkering with these parameters is more likely to yield substantial improvements than with other parameters of the AT-BBPSO algorithms, though this is not always the case. Appendix Appendix B: justifies these choices in an extended simulation study on several test functions.

In the AT-BBPSOxp variants, the improvement rate is in a certain sense poorly targeted — the rate is used to tune $\sigma(t)$ in (4), but 50% of the time $\theta_{ij}(t+1)$ will be set to $g_{ij}(t)$. So when $\boldsymbol{\theta}_i$ moves to a new personal best location, this may be because $\sigma(t)$ was set appropriately, or because it was forced to move to $\boldsymbol{g}_i$'s coordinate on several dimensions, or both. Despite this, tuning $\sigma(t)$ in the fashion described above still seems to perform well, which can be seen in Appendix Appendix B:.

2.2 Adaptively tuned PSO

In AT-BBPSO variants, the parameter $\sigma(t)$ partially controls the effective size of the swarm's search area, and we increase or decrease $\sigma(t)$ and consequently the search area depending on how much of the swarm is finding new personal best locations. In standard PSO there is no direct analogue to $\sigma(t)$, though the inertia parameter, ω in (1), is related. It controls the effective size of the swarm's search area by controlling how the magnitude of the velocities evolve over time — larger values of $\omega(t)$ allow for larger magnitude velocities in future periods.

In AT-BBPSO we use an analogy with tuning a random walk Metropolis-Hastings MCMC algorithm in order to build intuition about how to tune $\sigma(t)$. The analogy is much weaker in this case; nonetheless, we allow $\omega(t)$ to be time-varying and use the same mechanism in order to tune it as we did for $\sigma(t)$.

The idea of time-varying $\omega(t)$ has been in the PSO literature for some time. An early suggestion was to set $\omega(0) = 0.9$ and deterministically decrease it until it reaches $\omega(T) = 0.4$ after the maximum number of iterations allowed (Eberhart and Shi, 2000). In particular, Tuppadung and Kurutach (2011) suggest defining $\omega(t)$ via the parameterized inertia weight function

$$\omega(t) = \frac{1}{1 + \left(\frac{t}{\alpha}\right)^{\beta}} \tag{5}$$

where α and β are user-defined parameters. Roughly, α controls how low $\omega(t)$ can go and β controls how fast it gets there, so α and β can be thought of as intercept and slope parameters respectively. The suggestion in Tuppadung and Kurutach (2011) is to set α to a small fraction of the total amount of iterations in which the algorithm is allowed to run (e.g., 10% or 20%), and set β between one and four.

This approach tends to improve on standard PSO if $\omega(t)$'s progression is set appropriately, but it invariably makes using PSO more difficult for the average user. Additionally, depending on the problem, it may be more useful to let the swarm explore the space for more or less iterations, necessitating different progressions of $\omega(t)$. A priori it may not be clear exactly which approach is best for any given problem, so an automatic method is desirable. Adaptively tuned PSO (AT-PSO) is just that — it provides an automatic method to adjust the value of $\omega(t)$ depending on local information obtained by the particle swarm. Formally,

the AT-PSO updating equations are as follows:

$$
\begin{aligned}
\log \omega(t+1) &= \log \omega(t) + c \times sgn(R(t) - R^*), \\
\boldsymbol{v}_i(t+1) &= \omega(t+1)\boldsymbol{v}_i(t) + \phi_1 \boldsymbol{r}_{1i}(t) \circ \{\boldsymbol{p}_i(t) - \boldsymbol{\theta}_i(t)\} + \phi_2 \boldsymbol{r}_{2i}(t) \circ \{\boldsymbol{g}_i(t) - \boldsymbol{\theta}_i(t)\}, \\
\boldsymbol{\theta}_i(t+1) &= \boldsymbol{\theta}_i(t) + \boldsymbol{v}_i(t+1), \\
\boldsymbol{p}_i(t+1) &= \begin{cases} \boldsymbol{p}_i(t) & \text{if } \; Q(\boldsymbol{p}_i(t)) \geq Q(\boldsymbol{\theta}_i(t+1)) \\ \boldsymbol{\theta}_i(t+1) & \text{otherwise,} \end{cases} \\
\boldsymbol{g}_i(t+1) &= \arg \max_{\{\boldsymbol{p}_j(t+1) | j \in \mathcal{N}_i\}} Q(\boldsymbol{p}_j(t+1))
\end{aligned} \tag{6}
$$

where $R(t)$ is the improvement rate of the swarm in iteration t, R^* is the target improvement rate, and c controls how much $R(t)$ changes on a per iteration basis. For AT-BBPSO we used an analogy with random walk Metropolis-Hastings algorithms to suggest that a good value for the target improvement rate is smaller than 0.5 but not too small, and simulations in Appendix Appendix B: corroborate this and suggests that $R^* = 0.3$ or 0.5 are good values in many problems. The analogy does not apply so cleanly here though we find in Appendix Appendix B: that $R^* = 0.3$ or 0.5 still seems to work well for AT-PSO. In Appendix Appendix B: we speculate that lower values of c will result in higher inertias for longer, and as a result improve AT-PSO algorithms when more initial exploration of the search space is desirable. We use $c = 0.1$ as a default value, but in simulations (not reported here) we find that the gains from optimizing c appear to be small, though a very small or very large value can cause the algorithm to perform poorly.

A major strength of AT-PSO is that, unlike DI-PSO, it can increase $\omega(t)$ when information from the swarm suggests there is an unexplored high value region of the space — when too much of the swarm is improving on their personal best locations AT-PSO increases $\omega(t)$ until velocities start increasing, the swarm starts exploring a larger amount of the nearby space, and more of the particles fail to find improvements on their personal best. This mechanism provides a way for the swarm to adapt its behavior on the fly to the local conditions,

though like many other methods of improving PSO algorithms, it may cause premature convergence in multimodal problems.

Appendix Appendix B: contains an extended simulation study comparing a variety of these PSO and BBPSO algorithms on a suite of test functions and motivates the following recommendations. When the goal is convergence, AT-PSO with $R^* = 0.3$ or 0.5 and ring-1 or ring-3 neighborhoods tends to work the best, though in problems where convergence is difficult AT-BBPSOxp with $R^* = 0.3$ or 0.5 and ring-1 or ring-3 neighborhoods will often find better evaluations of the objective function while they and alternative PSO algorithms fail to converge. This feature of AT-PSO suggests a hybrid strategy using a Stage 1 optimization algorithm to get close to the global optimum, then using AT-PSO in Stage 2 in order to obtain convergence. We explore this strategy in Section 5 for several PSO algorithms in two statistical examples where the goal is to find the posterior mode for the Laplace approximation to the posterior. There we use the Broyden-Fletcher-Goldfarb-Shannon (BFGS) algorithm in Stage 1.

3 PSO Assisted Independent Metropolis-Hastings

Next we turn to using PSO algorithms to help construct MCMC algorithms for sampling from a posterior distribution. Let $\boldsymbol{\theta}$ be the model parameter and $p(\boldsymbol{\theta}|\boldsymbol{y}_{1:n})$ be its posterior distribution, available up to a normalizing constant. Further let $\boldsymbol{\theta}_n^*$ denote the posterior mode of $p(\boldsymbol{\theta}|\boldsymbol{y}_{1:n})$ and let $\boldsymbol{H}_n(\boldsymbol{\theta}_n^*)$ denote the Hessian matrix of $\log p(\boldsymbol{\theta}|\boldsymbol{y}_{1:n})$ evaluated at $\boldsymbol{\theta}_n^*$. Then under suitable regularity conditions (Schervish, 1997, Sections 7.4.2 and 7.4.3) $\boldsymbol{\theta}$'s posterior distribution is asymptotically normal. So, for a fixed but large value of n, $\boldsymbol{\theta}|\boldsymbol{y}_{1:n} \overset{a}{\sim} N(\boldsymbol{\theta}_n^*, -\boldsymbol{H}_n^{-1}(\boldsymbol{\theta}_n^*))$ where the notation $\overset{a}{\sim}$ means "approximately distributed as."

The Laplace approximation to the posterior can be used to create a proposal distribution for an independent Metropolis MCMC algorithm, though a multivariate t-distribution is

usually substituted for the normal so that the proposal has fatter tails than the target posterior. Finding the posterior mode in closed form is typically impossible and standard numerical optimization algorithms perform poorly in high dimensional parameter spaces, but PSO can perform well in larger spaces. We use PSO to find the posterior mode, but other heuristic algorithms such as genetic algorithms (Goldberg and Holland, 1988) could also be used. Additionally, for independent Metropolis algorithms it is not important that we find the true posterior mode; what is important is that we are close enough that the resulting Metropolis algorithm has a high acceptance rate. However, as we shall see in Section 5, sometimes when the estimate is slightly off the mode the algorithm will still have a poor acceptance rate.

This sort of algorithm is more likely to be useful in the tall data setting, i.e. when the number of observations is large and much larger than the number of covariates, because the normal approximation should be fairly good and there is a reduced likelihood of an ill-behaved posterior density, e.g., a bimodal posterior distribution. Some parameters have no observations directly related to them, but instead define the distribution of some latent process. For these parameters it is crucial that there are enough realizations of the latent process in the model to ensure that the normal approximation works well for the higher level parameter. More formally, suppose the model can be written as $L(\boldsymbol{\theta}, \boldsymbol{\psi}|\boldsymbol{y}_{1:n}) = \prod_{i=1}^{n} p(\boldsymbol{y}_i|\boldsymbol{\psi}) \times p(\boldsymbol{\psi}|\boldsymbol{\theta})$. Here $\boldsymbol{y}_i$ may include response variables and covariates, $\boldsymbol{\psi}$ may include data model parameters and a latent process, and $\boldsymbol{\theta}$ includes all higher level parameters. As long as n is large and the dimension of $\boldsymbol{\psi}$ is small, the normal approximation should work well for $\boldsymbol{\psi}$. In order for the normal approximation to work well for $\boldsymbol{\theta}$ as well, the elements of $\boldsymbol{\psi}$ need to depend on $\boldsymbol{\theta}$ in such a way that borrows strength across elements of $\boldsymbol{\psi}$. For example $p(\boldsymbol{\psi}|\boldsymbol{\theta}) = \prod_{j=1}^{m} p(\psi_j|\boldsymbol{\theta})$ where m is larger than the dimension of $\boldsymbol{\theta}$. Many common models fit into this framework, for example, generalized linear mixed effect models where the number of random effects is relatively small and constant in the sample size and the covariance matrix of the ran-

dom effects is a function of a low dimensional parameter. Some examples of these types of models are provided in Section 4. While, strictly speaking, these are not necessary nor sufficient conditions for a PSO assisted Metropolis-Hastings algorithm to work well, they do serve as useful guidelines. For example, when the covariance matrix of the random effect vector in a generalized linear mixed model is fully parameterized we essentially have one observation to estimate it — the estimate of the full random effect vector. As a result, the normal approximation is very poor and as we demonstrate in Section 6, the resulting independent Metropolis-Hastings algorithm has poor acceptance rates. In some models analytically finding the Hessian may be complicated, but unfortunately this is necessary in higher dimensions because numerical methods to evaluate the Hessian will often fail or be prohibitively slow. The independent Metropolis-Hastings algorithm we use is standard and given by Algorithm 1:

Algorithm 1 (Independent Metropolis-Hastings with Laplace approximation) *Given a user-chosen degrees of freedom parameter df, an estimate of the posterior mode* $\boldsymbol{\theta}^*$ *and the Hessian* $\boldsymbol{H}^* \equiv \boldsymbol{H}(\boldsymbol{\theta}^*)$ *with* $\ell^* = \log p(\boldsymbol{\theta}^*|\boldsymbol{y}_{1:n})$ *and* $\boldsymbol{\Sigma}^* = (-\boldsymbol{H}^*)^{-1}$*, obtain* $\boldsymbol{\theta}^{(t+1)}$ *from* $\boldsymbol{\theta}^{(t)}$ *via:*

1. *Draw proposal* $\boldsymbol{\theta}^{(prop)} \sim t_{df}(\boldsymbol{\theta}^*, \boldsymbol{\Sigma}^*)$.
2. *Compute the log posterior* $\ell^{(prop)} = \log p(\boldsymbol{\theta}^{(prop)}|\boldsymbol{y}_{1:n})$.
3. *Compute the log acceptance ratio*

$$\log a = \ell^{(prop)} - \ell^{(t)} + \log t_{df}(\boldsymbol{\theta}^{(t)}; \boldsymbol{\theta}^*, \boldsymbol{\Sigma}^*) - \log t_{df}(\boldsymbol{\theta}^{(prop)}; \boldsymbol{\theta}^*, \boldsymbol{\Sigma}^*).$$

4. *Accept* $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(prop)}$ *and* $\ell^{(t+1)} = \ell^{(prop)}$ *with probability* $\min(a, 1)$*, otherwise set* $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)}$ *and* $\ell^{(t+1)} = \ell^{(t)}$.

Here we abuse notation slightly and define $t_{df}(\cdot; \boldsymbol{\theta}, \boldsymbol{\Sigma})$ as the multivariate t density function with location parameter and scale parameter $\boldsymbol{\Sigma}$. Simulation from a multivariate t density

can be accomplished by drawing $\omega \sim IG(df/2, df/2)$, then $\boldsymbol{x} \sim N(\boldsymbol{\theta}, \omega\boldsymbol{\Sigma})$, but note that if $\boldsymbol{t} \overset{iid}{\sim} t_{df}$ and $\boldsymbol{C}\boldsymbol{C}' = \boldsymbol{\Sigma}$, $\boldsymbol{\theta} + \boldsymbol{C}\boldsymbol{t}$ is *not* a draw from the $t_{df}(\boldsymbol{\theta}, \boldsymbol{\Sigma})$ distribution (Hofert, 2013).

Ideally *df* should be set small enough so that the tails of $t_{df}(\boldsymbol{\theta}; \boldsymbol{\theta}^*, \boldsymbol{\Sigma}^*)$ dominate those of $p(\boldsymbol{\theta}|\boldsymbol{y}_{1:n})$ in order to ensure that the Markov chain is uniformly ergodic — see Robert and Casella (2013, Theorem 7.8). Aside from that constraint, *df* can be set to optimize the acceptance rate of the algorithm and in practice the constraint can often be taken into account by monitoring convergence of the chain and adjusting *df* appropriately. [DOUBLE CHECK HOW MUCH YOU NEED TO PAY ATTENTION TO ERGODICITY] When we have a good proposal which dominates the tails of our target, it is tempting to use a rejection sampling algorithm in order to exactly sample from our target instead of constructing a Markov chain with Metropolis-Hastings. However, independent Metropolis-Hastings is at least as efficient as rejection sampling in the sense of requiring less draws from the proposal to achieve the same Monte Carlo standard error (Liu, 1996) and with a good proposal convergence is typically very fast.

Often only part of $\boldsymbol{\theta}$ is approximately normal in the posterior. As long as the other part has a tractable conditional posterior we can adapt Algorithm 1 into an IMH within Gibbs (IMHwG) sampler.

Algorithm 2 (Independent Metropolis-Hastings within Gibbs with Laplace approximation) *Suppose* $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ *and that* $p(\boldsymbol{\theta}_2|\boldsymbol{\theta}_1, \boldsymbol{y}_{1:n})$ *can easily be drawn from. Given a user-chosen degrees of freedom parameter df, an estimate of the posterior mode* $\boldsymbol{\theta}^* = (\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*)$*, the Hessian* $\boldsymbol{H}^* \equiv \boldsymbol{H}(\boldsymbol{\theta}^*)$*, and*

$$\boldsymbol{\Sigma}^* = (-\boldsymbol{H}^*)^{-1} = \begin{bmatrix} \boldsymbol{\Sigma}_{11}^* & \boldsymbol{\Sigma}_{12}^* \\ \boldsymbol{\Sigma}_{21}^* & \boldsymbol{\Sigma}_{22}^* \end{bmatrix},$$

with $\ell^* = \log p(\boldsymbol{\theta}^*|\boldsymbol{y}_{1:n})$*, obtain* $\boldsymbol{\theta}^{(t+1)}$ *from* $\boldsymbol{\theta}^{(t)}$ *via:*

1. *Draw* $\boldsymbol{\theta}_2^{(t+1)} \sim p(\boldsymbol{\theta}_2|\boldsymbol{\theta}_1^{(t)}, \boldsymbol{y}_{1:n})$.

2. *Draw proposal* $\boldsymbol{\theta}_1^{(prop)} \sim t_{df}(\widetilde{\boldsymbol{\theta}}_1, \widetilde{\boldsymbol{\Sigma}}_{11})$*, where* $\widetilde{\boldsymbol{\theta}}_1 = \boldsymbol{\theta}_1^* + \boldsymbol{\Sigma}_{12}^*(\boldsymbol{\Sigma}_{22}^*)^{-1}(\boldsymbol{\theta}_2^{(t+1)} - \boldsymbol{\theta}_2^*)$ *and* $\widetilde{\boldsymbol{\Sigma}}_{11} = \boldsymbol{\Sigma}_{11}^* - \boldsymbol{\Sigma}_{12}^*(\boldsymbol{\Sigma}_{22}^*)^{-1}\boldsymbol{\Sigma}_{21}^*$.

3. *Compute the log acceptance ratio*

$$\begin{aligned}\log a = &\log p(\boldsymbol{\theta}_1^{(prop)}, \boldsymbol{\theta}_2^{(t+1)}|\boldsymbol{y}_{1:n}) - \log p(\boldsymbol{\theta}_1^{(t)}, \boldsymbol{\theta}_2^{(t+1)}|\boldsymbol{y}_{1:n}) \\ &+ \log t_{df}(\boldsymbol{\theta}_1^{(t)}; \boldsymbol{\theta}_1^*, \boldsymbol{H}_1^*) - \log t_{df}(\boldsymbol{\theta}_1^{(prop)}; \boldsymbol{\theta}_1^*, \boldsymbol{H}_1^*).\end{aligned}$$

4. *Accept* $\boldsymbol{\theta}_1^{(t+1)} = \boldsymbol{\theta}_1^{(prop)}$ *with probability* $\min(a, 1)$*, otherwise set* $\boldsymbol{\theta}_1^{(t+1)} = \boldsymbol{\theta}_1^{(t)}$.

This algorithm approximates the conditional posterior of $\boldsymbol{\theta}_1$ given $\boldsymbol{\theta}_2$ with a the conditional distribution of $\boldsymbol{\theta}_1$ given $\boldsymbol{\theta}_2$ implied by the normal approximation to the joint posterior of $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$. Ideally, we would directly approximate the conditional posterior of $\boldsymbol{\theta}_1$. However, this is usually too computationally expensive because it would require running an optimization algorithm every iteration of the MCMC algorithm in order to find the conditional mode. This approach will often yield higher acceptance rates, but it is only attractive when the conditional mode is available in closed form or cheaply available through other means. In other words, this is attractive in precisely the situations where the optimization problem is too easy for PSO to be advantageous.

4 Generalized linear mixed model applications

Generalized linear mixed models with latent Gaussian processes (LGP) provide a plethora of examples where PSO assisted IMH algorithms are attractive for MCMC. That latent parameters are Gaussian distributed is a crucial feature that increases the quality of the normal approximation to the posterior. We will conceptualize our model class using the strategy of Berliner (1996) and Wikle et al. (2003), that is hierarchically with a data model conditional on parameters and a latent process, a process model conditional on parameters, and

finally a parameter model. Let $\boldsymbol{Z} = (Z_1, Z_2, \ldots, Z_n)$ denote a vector of response variables. We assume a conditionally independent data model given a vector of location parameters $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_n)$ and a common dispersion parameter ϕ, i.e. $Z_i \overset{ind}{\sim} f(Z|\mu_i, \phi)$ where $f(z|\mu, \phi)$ is some known family of density functions indexed by (μ, ϕ). The mean parameters are a known function g of a LGP, so $g^{-1}(\boldsymbol{\mu}) = \boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{S}\boldsymbol{\delta}$. Here $\boldsymbol{X}'\boldsymbol{\beta}$ represents fixed effects and $\boldsymbol{S}'\boldsymbol{\delta}$ represents random effects, where $\boldsymbol{X}$ is a known $n \times p$ matrix, $\boldsymbol{S}$ is a known $n \times r$ matrix, $\boldsymbol{\beta}$ is an unknown p-dimensional vector and $\boldsymbol{\delta}$ is an unknown r-dimensional vector. $\boldsymbol{\delta}$ is further modeled as Gaussian with mean zero and a covariance matrix $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ with a structure appropriate to the specific problem. Finally a prior on $(\phi, \boldsymbol{\beta}, \boldsymbol{\theta})'$ serves as the parameter model. The model can be generalized to multivariate Z_i and multivariate Y_i, but we omit this complication here. As long as the conditions mentioned in Section 3 are satisfied, the Laplace approximation is likely to work well, PSO is likely to find the mode efficiently, and Algorithm 1 is likely to yield efficient MCMC. When Algorithm 1 fails to have high enough acceptance rates specifically because the Laplace approximation is poor for only some of the mode's parameters, Algorithm 2 will often still work — e.g., when $\boldsymbol{\Sigma}$ is fully parameterized. The next two subsections describe two examples of this class of models. In Section 4.1 we describe several classes of reduced rank spatial models for 2014 American Community Survey (ACS) 5-year period estimates of county populations. In Section 4.2 we describe two models for predicting the result of the 1988 presidential election based on a series of national polls.

4.1 Spatially modeling county population estimates

The American Community Survey (ACS) provides 5-year period estimates of county populations as recently as 2014. In 2014 there were 3,142 counties in the United States, including the District of Columbia and counties in Alaska and Hawaii. We use two separate data

models in order to illustrate when the normal approximation works well. The first is a Poisson data model, i.e., $Z_i \sim \text{Poisson}(\lambda_i)$ where $\lambda_i = \exp(Y_i)$. Through visual inspection of a histogram, it was determined that, on the log scale, county populations look approximately normally distributed. So, our second data model is $\log Z_i \sim N(\mu_i, \phi^2)$ where $\mu_i = Y_i$.

The process model in both cases is a reduced rank spatial model $Y = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{S}\boldsymbol{\delta}$, where $\boldsymbol{X}\boldsymbol{\beta}$ represents the process mean at each county and $\boldsymbol{S}\boldsymbol{\delta}$ implies the spatial correlation across counties. The spatial correlation term consists of a set of r basis functions evaluated at each of the $n = 3,142$ counties, denoted by the $n \times r$ matrix $\boldsymbol{S}$, and a common random effect $\boldsymbol{\delta}$. We assume that $\boldsymbol{\delta}$ is r-dimensional with $r \ll n$ so that the model is reduced rank. Any set of spatial basis functions could be used for $\boldsymbol{S}$ but we use the Moran's I (MI) basis set, described below (see Hughes and Haran (2013), Porter et al. (2015), Bradley et al. (2015) and references therein for additional discussion). Another possibility is to define a reduced rank model for a point-level spatial process using a basis function expansion and compute the implied set of basis functions for each of the census tracts by integrating the point level basis functions appropriately. See Sections 2.1, 3.1, and 4 of Bradley et al. (2016) for details.

The MI basis functions are defined through the orthogonal projection matrix $\boldsymbol{P_X} = \boldsymbol{X}(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'$. Let $\boldsymbol{A}$ denote the binary adjacency matrix with $a_{ij} = 1$ if counties i and j are neighbors, $a_{ij} = 0$ otherwise, and $a_{ii} = 0$ along the diagonal, and define the MI operator $\boldsymbol{G}$ as

$$\boldsymbol{G} = (\boldsymbol{I}_n - \boldsymbol{P_X})\boldsymbol{A}(\boldsymbol{I}_n - \boldsymbol{P_X})$$

where $\boldsymbol{I}_n$ is the $n \times n$ identity matrix. The spectral decomposition of $\boldsymbol{G}$ is $\boldsymbol{G} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}'$. To use a reduced rank version of the MI basis functions we truncate the basis function expansion and take $\boldsymbol{S}$ to be the $n \times r$ matrix formed by the r columns of $\boldsymbol{\Phi}$ corresponding to the largest r eigenvalues of $\boldsymbol{G}$. The random effect $\boldsymbol{\delta}$ is further modeled as $\boldsymbol{\delta} \sim N(\boldsymbol{0}_r, \boldsymbol{\Sigma}(\boldsymbol{\theta}))$ where $\boldsymbol{0}_r$ denotes an r-dimensional vector of zeroes and $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ is an unknown covariance matrix. The

covariance matrix of $\boldsymbol{S\delta}$ is then $\boldsymbol{S\Sigma}(\boldsymbol{\theta})\boldsymbol{S}'$.

The process model depends on choices for $\boldsymbol{X}$ and r. In practice r can be chosen using a sensitivity analysis. Since our goal is to illustrate computational methods, we will elide choosing r in a principled way and instead use several values for r in order to illustrate when the parameter space becomes too high dimensional for our method to be advantageous. For simplicity we choose an intercept only model, but all derivations for the model will assume that $\boldsymbol{X}$ is $n \times p$.

Finally, we consider two distinct parameterizations of $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ — the iid parameterization, and the full parameterization. In the iid parameterization the prior for $\boldsymbol{\Sigma}$ is $\sigma^2 \sim IG(a_\sigma, b_\sigma)$ where $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}_r$, while in the full parameterization we assume that $\boldsymbol{\Sigma} \sim IW(d, \boldsymbol{E})$. The prior for $\boldsymbol{\beta}$ in all models is $\boldsymbol{\beta} \sim N(\boldsymbol{b}, v^2 \boldsymbol{I}_p)$, and the lognormal models the prior for ϕ^2 is $\phi^2 \sim IG(a_\phi, b_\phi)$. We assume that the parameters $\boldsymbol{\beta}$, $\boldsymbol{\Sigma}$, and when applicable ϕ^2 are mutually independent in the prior. For our examples we assume that $a_\sigma = a_\phi = b_\sigma = b_\phi = 1$, $\boldsymbol{b} = \boldsymbol{0}_p$, $v = 10$, $d = r + 1$ and $\boldsymbol{E} = \boldsymbol{I}_p$. Often a more complicated prior is appropriate on variance or covariance matrix parameters so that the marginal posterior is not sensitive to arbitrary choices in the prior. We use these conditionally conjugate priors because they allow for a fair comparison between MCMC algorithms — most alternatives will complicate Gibbs samplers with extra steps or necessitate Metropolis steps making Algorithm 1 relatively more attractive.

Between the two possible parameterizations of $\boldsymbol{\Sigma}$ and the two choices for the data model — Poisson versus Lognormal — we consider four possible classes of models. Then for the

models with iid random effects the posterior distributions can be written as

$$p(\boldsymbol{\beta},\sigma^2,\boldsymbol{\delta},\phi^2|\boldsymbol{z},\boldsymbol{X},\boldsymbol{S}) \propto (\phi^2)^{-n/2-a_\phi-1}\exp\left[-\frac{1}{\phi^2}\left\{\frac{(\log\boldsymbol{z}-\boldsymbol{y})'(\log\boldsymbol{z}-\boldsymbol{y})}{2}+b_\phi\right\}\right]$$
$$\times(\sigma^2)^{-\frac{r}{2}-a_\sigma-1}\exp\left\{-\frac{1}{\sigma^2}\left(\frac{\boldsymbol{\delta}'\boldsymbol{\delta}}{2}+b_\sigma\right)\right\}\exp\left\{-\frac{(\boldsymbol{\beta}-\boldsymbol{b})'(\boldsymbol{\beta}-\boldsymbol{b})}{2v}\right\} \quad \text{(iid lognormal)}, \tag{7}$$

$$p(\boldsymbol{\beta},\sigma^2,\boldsymbol{\delta}|\boldsymbol{z},\boldsymbol{X},\boldsymbol{S}) \propto \prod_{i=1}^{n}\frac{\exp\{-\exp(y_i)\}\exp(y_i z_i)}{z_i!}$$
$$\times(\sigma^2)^{-\frac{r}{2}-a_\sigma-1}\exp\left\{-\frac{1}{\sigma^2}\left(\frac{\boldsymbol{\delta}'\boldsymbol{\delta}}{2}+b_\sigma\right)\right\}\exp\left\{-\frac{(\boldsymbol{\beta}-\boldsymbol{b})'(\boldsymbol{\beta}-\boldsymbol{b})}{2v}\right\} \quad \text{(iid Poisson)}. \tag{8}$$

For fully parameterized $\boldsymbol{\Sigma}$ we write the posteriors in terms of the precision matrix $\boldsymbol{\Omega}=\boldsymbol{\Sigma}^{-1}$, yielding

$$p(\boldsymbol{\beta},\boldsymbol{\Omega},\boldsymbol{\delta},\phi^2|\boldsymbol{z},\boldsymbol{X},\boldsymbol{S}) \propto (\phi^2)^{-n/2-a_\phi-1}\exp\left\{-\frac{1}{\phi^2}\left(\frac{(\log\boldsymbol{z}-\boldsymbol{y})'(\log\boldsymbol{z}-\boldsymbol{y})}{2}+b_\phi\right)\right\}$$
$$\times|\boldsymbol{\Omega}|^{(d-r)/2}\exp\left\{-\frac{1}{2}\left(\boldsymbol{\delta}'\boldsymbol{\Omega}\boldsymbol{\delta}+\frac{(\boldsymbol{\beta}-\boldsymbol{b})'(\boldsymbol{\beta}-\boldsymbol{b})}{v}+\mathrm{tr}(\boldsymbol{E}\boldsymbol{\Omega})\right)\right\} \quad \text{(full lognormal)}, \tag{9}$$

$$p(\boldsymbol{\beta},\boldsymbol{\Omega},\boldsymbol{\delta}|\boldsymbol{z},\boldsymbol{X},\boldsymbol{S}) \propto \prod_{i=1}^{n}\frac{\exp\{-\exp(y_i)\}\exp(y_i z_i)}{z_i!}$$
$$\times|\boldsymbol{\Omega}|^{(d-r)/2}\exp\left\{-\frac{1}{2}\left(\boldsymbol{\delta}'\boldsymbol{\Omega}\boldsymbol{\delta}+\frac{(\boldsymbol{\beta}-\boldsymbol{b})'(\boldsymbol{\beta}-\boldsymbol{b})}{v}+\mathrm{tr}(\boldsymbol{E}\boldsymbol{\Omega})\right)\right\} \quad \text{(full Poisson)} \tag{10}$$

where $\mathrm{tr}(\cdot)$ denotes the trace operator. For some MCMC algorithms it will be easier to work with the Cholesky decomposition of $\boldsymbol{\Omega}$ given by $\boldsymbol{L}\boldsymbol{L}'=\boldsymbol{\Omega}$ where $\boldsymbol{L}$ is lower triangular. In practice it is often more convenient to put the prior distribution directly on $\boldsymbol{L}$ rather than on $\boldsymbol{\Omega}$ or $\boldsymbol{\Omega}^{-1}$ and solving for the Jacobian, but this depends in part on which MCMC algorithm is used to fit the model. So while we use the prior distribution on $\boldsymbol{L}$ implied by a Wishart prior on $\boldsymbol{\Omega}$, in practice it is advantageous to use one of the priors suggested by Chen and Dunson (2003) or Frühwirth-Schnatter and Tüchler (2008). We allow the diagonal entries of $\boldsymbol{L}$ to be negative in the independent Metropolis-Hastings algorithms in order to facilitate MCMC, so $\boldsymbol{L}$ is not strictly speaking a Cholesky decomposition. The determinant

of the Jacobian is the same in both cases up to a proportionality constant. The signs of the elements of $\boldsymbol{L}$ are not identified, therefore care needs to be taken when interpreting the results of MCMC. Transforming back to the precision matrix in a post processing step is sufficient. Let ℓ_{ij} denote the (i,j)th element of $\boldsymbol{L}$. Then the Jacobian of $\boldsymbol{\Omega} \to \boldsymbol{L}$ is given by

$$|J(\boldsymbol{\Omega} \to \boldsymbol{L})| \propto \prod_{k=1}^{r} |\ell_{kk}|^{r+1-k}$$

where $\boldsymbol{\Omega}$ is $r \times r$. Under this parameterization the full posteriors can be written as

$$p(\boldsymbol{\beta},\boldsymbol{L},\boldsymbol{\delta},\phi^2|\boldsymbol{z},\boldsymbol{X},\boldsymbol{S}) \propto (\phi^2)^{-n/2-a_\phi-1} \exp\left[-\frac{1}{\phi^2}\left\{\frac{(\log \boldsymbol{z}-\boldsymbol{y})'(\log \boldsymbol{z}-\boldsymbol{y})}{2}+b_\phi\right\}\right] \prod_{k=1}^{r} (\ell_{kk}^2)^{(d-k+1)/2}$$
$$\times \exp\left[-\frac{1}{2}\left\{\boldsymbol{\delta}'\boldsymbol{L}\boldsymbol{L}'\boldsymbol{\delta}+\frac{(\boldsymbol{\beta}-\boldsymbol{b})'(\boldsymbol{\beta}-\boldsymbol{b})}{v}+\mathrm{tr}(\boldsymbol{E}\boldsymbol{L}\boldsymbol{L}')\right\}\right] \qquad \text{(full lognormal)}, \tag{11}$$

$$p(\boldsymbol{\beta},\boldsymbol{L},\boldsymbol{\delta}|\boldsymbol{z},\boldsymbol{X},\boldsymbol{S}) \propto \prod_{i=1}^{n} \frac{\exp\{-\exp(y_i)\}\exp(y_i z_i)}{z_i!} \times \prod_{k=1}^{r} (\ell_{kk}^2)^{(d-k+1)/2}$$
$$\times \exp\left[-\frac{1}{2}\left\{\boldsymbol{\delta}'\boldsymbol{L}\boldsymbol{L}'\boldsymbol{\delta}+\frac{(\boldsymbol{\beta}-\boldsymbol{b})'(\boldsymbol{\beta}-\boldsymbol{b})}{v}+\mathrm{tr}(\boldsymbol{E}\boldsymbol{L}\boldsymbol{L}')\right\}\right] \qquad \text{(full Poisson)}. \tag{12}$$

In Appendix A we derive the Hessian for the fully parameterized Poisson model. The other models are analogous, though the variances in the iid models and in the lognormal models should be transformed to the log scale first.

In Section 6 we consider several Gibbs sampling algorithms which draw the covariance matrix parameter $\boldsymbol{\theta}$ from its full conditional distribution. When the covariance matrix is fully parameterized the full conditional distribution of the precision matrix $\boldsymbol{\Omega}$ is Wishart, that is $\boldsymbol{\Omega} \sim W(\widetilde{d}, \widetilde{\boldsymbol{E}})$. This draw is usually accomplished via the Bartlett decomposition (Smith and Hocking, 1972). Let $\widetilde{\boldsymbol{C}}$ be the lower triangular Cholesky decomposition of $\widetilde{\boldsymbol{E}}$. Then let $\boldsymbol{A}$ be an $r \times r$ random lower triangular matrix with independent elements $\{a_{ij} : 0 < i \leq j \leq r\}$ where $a_{ii} \sim \sqrt{\chi^2_{\widetilde{d}-i+1}}$ for $i = 1, 2, \ldots, r$ and $a_{ij} \sim N(0,1)$ for $0 < i \leq j \leq r$. Then $\boldsymbol{\Omega} = \boldsymbol{L}\boldsymbol{L}' \sim W(\widetilde{d}, \widetilde{\boldsymbol{E}})$ where $\boldsymbol{L} = \widetilde{\boldsymbol{C}}\boldsymbol{A}$. In the process of drawing $\boldsymbol{\Omega}$ we must first draw $\boldsymbol{L}$, so we construct our Gibbs samplers in terms of $\boldsymbol{L}$ instead of $\boldsymbol{\Omega}$.

4.2 Predicting the 1988 presidential election

Gelman and Hill (2006, Chapter 14) describes a model used to predict state-level opinions about the 1988 presidential candidates from national polls in order to predict the outcome of the election. They model the responses to a series of seven polls conducted by CBS News during the week before the 1988 presidential election. The variable of interest is binary: $Z_i = 1$ if the ith respondent said they supported the Republican candidate and $Z_i = 0$ if they said they supported the Democratic candidate, with undecideds being excluded. Focusing on the last poll, they ultimately estimate a logistic regression model with fixed effects for race (whether the respondent was African American or not), sex, and race×sex, and random effects for four age categories, four education categories, and 16 age×education categories, as well as for the respondent's state of residence (including District of Columbia). The mean of the state random effect distribution is one of five region random effects plus the proportion of the state that voted republican in the last election times a slope coefficient.

We reduce the size of this model somewhat by omitting the age and education random effects, but keeping the age×education interaction terms — so the age×education random effects now represent the random effects for each age×education category. The single poll data model is

$$P(Z_i = 1) = \theta_i, \quad \theta_i = \exp(Y_i)/\{1 + \exp(Y_i)\},$$
$$Y_i = \beta_0 + f_i\beta_f + b_i\beta_b + f_i b_i \beta_{fb} + \alpha_{ae}[ae_i] + \alpha_s[s_i] \quad \text{(single poll data model)}, \quad (13)$$

where f_i indicates whether respondent i identified as female, b_i indicates whether respondent i identified as African American, ae_i indicates respondent i's age×education category, and s_i indicates respondent i's state of residence. Here we use $\alpha_s[k]$ to denote the kth element of the vector $\boldsymbol{\alpha}_s$, so $\boldsymbol{\alpha}_{ae}$ contains 16 elements, and $\boldsymbol{\alpha}_s$ contains 51 elements (50 states plus

the District of Columbia). The single poll process model is

$$\begin{aligned}
&\alpha_s[k] \overset{ind}{\sim} N(\alpha_r[r_k] + prev_k\beta_{prev}, \sigma_s^2) \text{ for } k = 1, 2, \ldots, 51, \\
&\alpha_{ae}[k] \overset{iid}{\sim} N(0, \sigma_{ae}^2) \text{ for } k = 1, 2, \ldots, 16, \\
&\alpha_r[k] \overset{iid}{\sim} N(0, \sigma_r^2) \text{ for } k = 1, 2, \ldots, 5 \quad \text{(single poll process model)},
\end{aligned} \tag{14}$$

where $\alpha_r[r_k]$ denotes the region containing state k and $prev_k$ denotes the average vote share for the Republicans in the previous three presidential elections. This model expands the class of models discussed at the beginning of this section by allowing the mean of $\boldsymbol{\delta}$ to depend on random effects that are further modeled. Adding a level to the hierarchy does not fundamentally change the applicability of PSO assisted MCMC algorithms, so long as the parameter space is still not too large for PSO to be feasible and the normal approximation is reasonable for the additional parameters.

The last poll had 2,015 respondents, but together all seven polls have 11,566 respondents. Using each poll with a minimal number of additional parameters to account for poll to poll variability should increase the quality of the model and result in a posterior with a better Laplace approximation. We analyze a model for all of the polls using the following data model

$$\begin{aligned}
&P(Z_i = 1) = \theta_i, \quad \theta_i = \exp(Y_i)/\{1 + \exp(Y_i)\}, \\
&\qquad Y_i = \beta_0 + f_i\beta_f + b_i\beta_b + f_ib_i\beta_{fb} + \alpha_{ae}[ae_i] + \alpha_s[s_i] + \alpha_p[p_i] \quad \text{(all polls data model)},
\end{aligned} \tag{15}$$

where p_i denotes which poll respondent i was surveyed in. The process model is given by

$$\begin{aligned}
&\alpha_s[k] \overset{ind}{\sim} N(\alpha_r[r_k] + prev_k\beta_{prev}, \sigma_s^2) \text{ for } k = 1, 2, \ldots, 51, \\
&\alpha_{ae}[k] \overset{iid}{\sim} N(0, \sigma_{ae}^2) \text{ for } k = 1, 2, \ldots, 16, \\
&\alpha_r[k] \overset{iid}{\sim} N(0, \sigma_r^2) \text{ for } k = 1, 2, \ldots, 5, \\
&\alpha_p[k] \overset{iid}{\sim} N(0, \sigma_p^2) \text{ for } k = 1, 2, \ldots, 7 \quad \text{(all polls process model)}.
\end{aligned} \tag{16}$$

In both models we assume each of the βs have independent $N(0, 1000)$ priors, and each of the σ^2s have $IG(1, 1)$ priors. Including the random effects the single poll model contains 80 parameters while the all polls model contains 89 parameters, so both models are large enough to be challenging for PSO and other optimization algorithms. Writing down the log posteriors and deriving the Hessians is straightforward but tedious for these models, so we omit these steps, though note that in both the PSO and IMH algorithms the variances should be transformed to the log scale.

5 PSO results for finding posterior modes

[I HAVE RESULTS USING 10 RANDOM EFFECTS iid MODELS AND 5 IN full; WORTH FINDING SPACE FOR?]

We conduct a simulation study using `R` (R Development Core Team, 2008) to compare various PSO algorithms at finding the posterior mode in each of the example models from Sections 4.1 and 4.2. Based on the results of Appendix Appendix B:, we limit the study to 7 PSO algorithms: standard PSO algorithm using parameter values suggested by Blum and Li (2008) and Clerc and Kennedy (2002) (PSO in Figures 1, 2, and [REFERENCE TO POLL BOX PLOT FIGURE]), the standard BBPSOxp-MC algorithm (BBPSO), DI-PSO with $\alpha = 0.2 \times n_{iter} = 200$ and $\beta = 1$ (DI-PSO), AT-PSO with $c = 0.1$ and either $R^* = 0.3$ or 0.5 (AT-PSO-0.3 and AT-PSO-0.5), and AT-BBPSOxp-MC with $df = 1$, $c = 0.1$, and either $R^* = 0.3$ or 0.5 (AT-BBPSO-0.3 and AT-BBPSO-0.5). Each algorithm was tried with one of two initializations. In the "BFGS" initialization, we first ran the BFGS algorithm using `R`'s `optim` function (R Development Core Team, 2008) until convergence using default settings to obtain an initial guess of the argmax, $\widehat{\boldsymbol{\theta}}$, then initialized the swarm with one particle at this initial guess and the rest uniformly in a length 2 hypercube centered on $\widehat{\boldsymbol{\theta}}$, i.e. $\boldsymbol{\theta}_1 = \widehat{\boldsymbol{\theta}}$ and $\theta_{ij} = \widehat{\theta}_j + U(-1, 1)$ for $i = 2, 3, \ldots, 50$ and $j = 1, 2, \ldots, n_{par}$ where n_{par} is the number of

parameters in the model and the $U(-1, 1)$ random variates are drawn independently. In the "no BFGS" initialization, each particle was initialized uniformly in a length 200 hypercube centered at zero, i.e. $\theta_{ij} \overset{iid}{\sim} U(-100, 100)$ for $i = 1, 2, \ldots, 50$ and $j = 1, 2, \ldots, n_{par}$. In addition, each algorithm was run using both the ring-1 and ring-3 neighborhood topologies, for a total of four combinations of initializations and neighborhoods, each for 20 replications of 1,000 iterations using 50 particles.

Figure 1 contains boxplots of the results of the simulations for the Poisson county population models of Section 4.1. Models with iid random effects had 30 random effects while models with fully correlated random effects had 15. Each box plot was created using the 10, 25, 50, 75, and 90 percentiles of the maximum value of the log posterior found in all 20 replications after 1,000 iterations. Figure 2 is similar, except for the lognormal models. Across all models we see that the ring-3 topology and BFGS initialization both improve each of the algorithms, especially in combination. In Appendix Appendix B: the ring-3 neighborhood performed the best on our suite of test functions, and the same seems to be true here. The BFGS initialization is cheap, taking essentially no time to compute, yet seems to drastically improve the quality of every PSO algorithms. In Appendix Appendix B: we speculated that the AT-PSO algorithms would benefit significantly from some sort of stage one optimization, and our results confirm this for all of the algorithms. Another lesson from these boxplots is that the AT-PSO and AT-BBPSO algorithms tend to do the best, especially when $R^* = 0.5$ in both cases.

[PARAGRAPH OR TWO AND PLOTS FOR THE ELECTION MODELS]

For our purposes, the PSO algorithms are useful only insofar as they allow us to construct IMH and IMHwG algorithms. So we conduct another simulation study to see when each algorithm gets close enough to the posterior mode for the IMH and IMHwG algorithms to have high acceptance rates. To this end we conduct another simulations study using the same 7 PSO algorithms from the previous study, except only using the ring-3 neighborhood

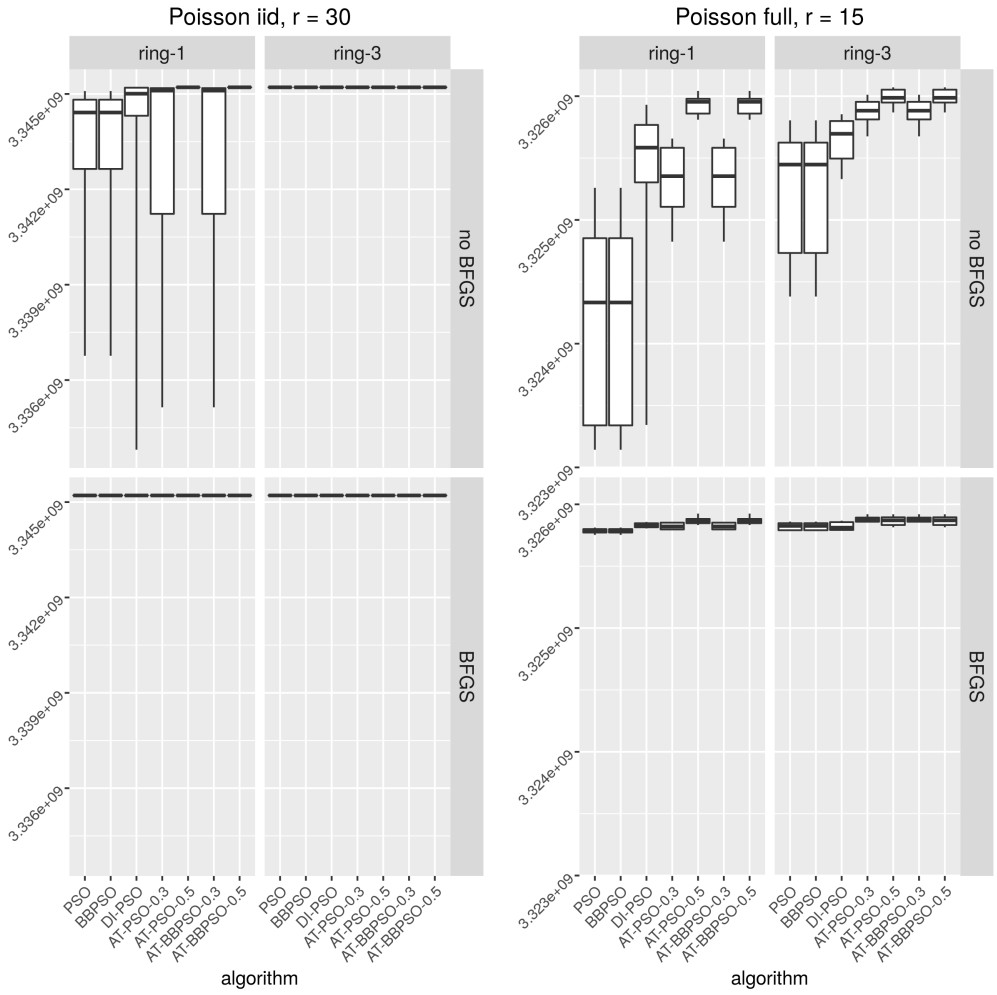

Figure 1: Boxplots of the maximum value of the log posterior found for Poisson models by random effect type, neighborhood topology, optimization initialization, and algorithm. Each box plot was created using the 10, 25, 50, 75, and 90 percentiles of the maximum found from 20 replications of 1,000 iterations with 50 particles for each factor combination.

topology and BFGS initialization. Next, we run Algorithms 1 and 2 after 0, 100, 500, 1,000, 1,500, and 2,000 iterations of the PSO algorithm and compute the acceptance rate after 1,000 iterations of the MCMC algorithm. Each MCMC algorithm is initialized at the PSO estimate of the posterior mode used to construct the Laplace approximation. When we run the IMH and IMHwG algorithms after 0 iterations of a PSO algorithm we still run the BFGS

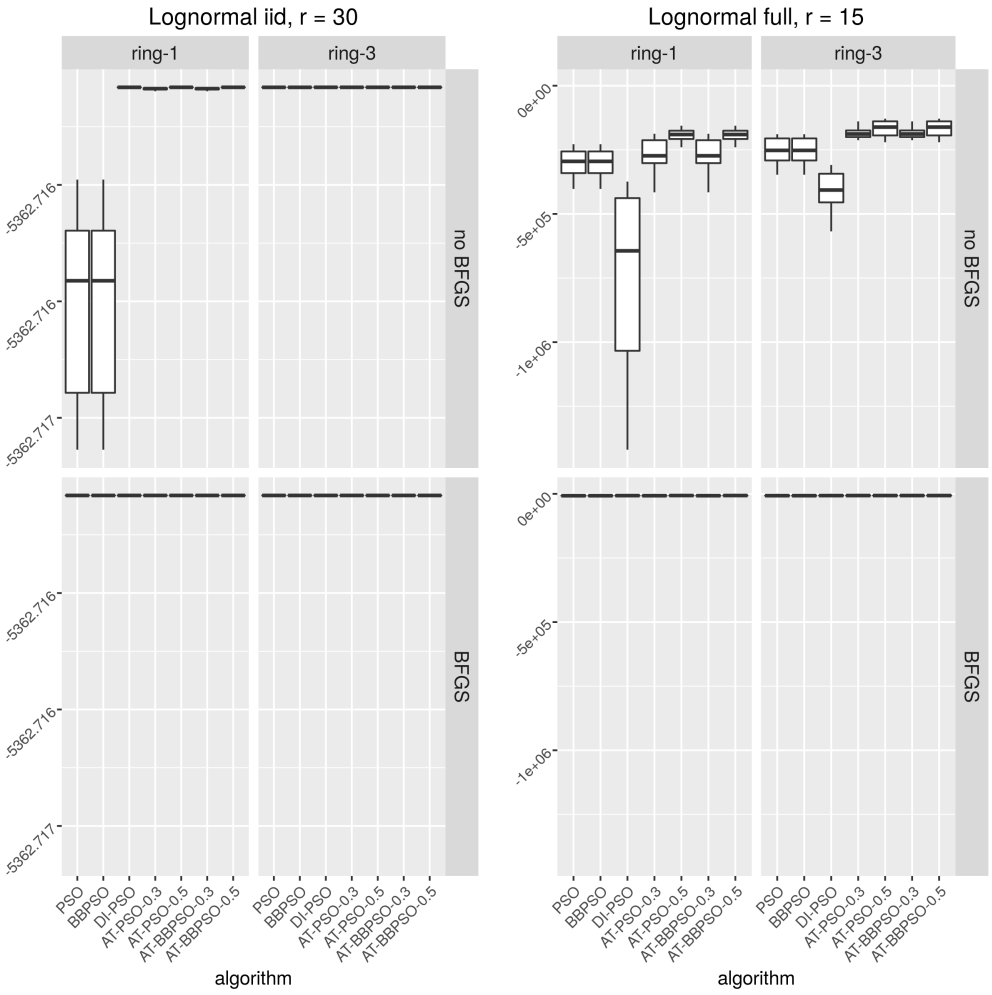

Figure 2: Boxplots of the maximum value of the log posterior found for lognormal models by random effect type, number of random effects, neighborhood topology, optimization initialization, and algorithm. Each box plot was created using the 10, 25, 50, 75, and 90 percentiles of the maximum found from 20 replications of 1,000 iterations with 50 particles for each factor combination.

initialization, so this serves as a control to see if running the PSO algorithm is necessary to get reasonable acceptance rates. Additionally, we run each IMH and IMHwG algorithm with different choices for the degrees of freedom parameter in the Laplace approximation.

Algorithm 1 applies straightforwardly to each of our models, though see Appendix A

for a detailed derivation of the Hessian for the county population models with a fully parameterized covariance matrix associated, i.e. from equations (11) and (12). To apply Algorithm 2 in the county population models, we draw $(\boldsymbol{\beta}, \boldsymbol{\delta})$ in the Metropolis step and either σ^2 or $\boldsymbol{L}$ in the conditionally conjugate step, depending on the model. The full conditional distribution of σ^2 in (7) and (8) is $IG(a_\sigma + r/2, b_\sigma + \boldsymbol{\delta}'\boldsymbol{\delta}/2)$. The full conditional distribution of $\boldsymbol{\Omega}$ in (9) and (10) is $W(r+1, (\boldsymbol{E} + \boldsymbol{\delta}\boldsymbol{\delta}')^{-1})$. Then a draw from the full conditional distribution of $\boldsymbol{L}$ can be obtained via the Bartlett decomposition as described at the end of Section 4.1. In the lognormal models the full conditional distribution of ϕ^2 is $IG(a_\phi + n/2, b_\phi + (\log \boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{S}\boldsymbol{\delta})'(\log \boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{S}\boldsymbol{\delta})/2)$.

In single poll model of Section 4.2, we draw $(\beta_0, \beta_f, \beta_b, \boldsymbol{\alpha}_s, \boldsymbol{\alpha}_{ae})$ in the Metropolis step, while in the all polls model we draw all of those parameters and additionally $\boldsymbol{\alpha}_p$ in the Metropolis step. Then for both models there are two additional Gibbs steps — one where each random effect variance is drawn, and one where $(\beta_{prev}, \boldsymbol{\alpha}_r)$ is drawn. These full conditionals are straightforward to derive, so we do not reproduce them here. Likewise, the Hessian is easy though tedious to derive, so we omit it as well.

[SIMULATION STUDY TO BE COMPLETED AND PUT HERE - WON'T TAKE VERY LONG, 1 DAY]

6 Comparison of MCMC techniques

Next we move to comparing various MCMC algorithms for both classes of models. The other MCMC algorithms we consider in this section are single move random walk Metropolis within Gibbs (RWwG), block random walk Metropolis within Gibbs (B-RWwG), Hamiltonian Monte Carlo as implemented by the Stan software (Carpenter et al., 2015) using the No U-turn Sampler (Homan and Gelman, 2014) and for the lognormal model for county populations, a two step Gibbs sampler (Gibbs). The last algorithm is only considered for the

lognormal model because only in that case are all of the full conditionals tractable. We use standard settings to fit each of the models in Stan, though minimal tweaking was required in some cases. In all of the Gibbs algorithms, the blocks are the same as in the IMHwG algorithms detailed in the previous section. Both the RWwG and B-RWwG algorithms are tuned during the burn-in using an adaptive method. Appendix A explains the details of the two random walk algorithms we use including how the adaptation was performed.

We compare these algorithms in Table [TO BE CONSTRUCTED] in terms of two measures: the minimum estimated effective sample size for all parameters in the model (n_{eff}, see Robert and Casella (2013, Section 12.3.5)), and time in seconds per n_{eff}. The effective sample size is the size of an iid sample which yields the same standard error for estimating the mean of some function of the parameters in our MCMC simulations, so we take the minimum estimated n_{eff} among all elements of the parameter vector and latent process.

[SIMULATIONS TO BE CONDUCTED - SHOULD TAKE A COUPLE DAYS ONCE THEY'RE RUNNING. FOR THE IMH AND IMHwG ALGORITHMS, THESE WILL ONLY USE ONE PSO ALGORITHM]

7 Discussion

[WILL NEED TO BE REWRITTEN IN LIGHT OF FORTHCOMING RESULTS]

In practice PSO-assisted Metropolis-Hastings algorithms are useful to the extent that the problem falls in something of a "sweet spot". The model must be complex enough that the posterior mode cannot be found analytically and that a fully conjugate Gibbs sampler is not available. The combined parameter space and latent space must be large enough that traditional numerical methods fail to find the posterior mode, but small enough that the heuristic PSO algorithms have some chance without being too costly. Finally, both model parameters and latent random variables must be approximately Gaussian in the posterior,

though there is some leeway for model parameters using the IMHwG algorithm. Under these conditions, using PSO to obtain the Laplace approximation for use as a proposal for IMH or IMH within Gibbs improves on other more commonly used approaches for MCMC. The comparison to Stan is also relevant — even though our PSO assisted IMH algorithms beat Stan in terms of total computational time for a desired level of precision for some of the models we considered, the difference will be small enough in many settings for most users to stick with the more general algorithm. [SOMETHING ABOUT THE ADVANTAGE BEING IN BIG / TALL DATA SITUATIONS - STAN REQUIRES A LOT OF LOG POSTERIOR EVALUATIONS]

Our approach is fairly hands off, so long as the conditions listed above appear to be satisfied. In that case the standard PSO algorithm with default parameter values and the ring-1 neighborhood usually does a great job of finding the posterior mode quickly relative to the most competitive alternatives. Alternatives such as the adaptively tuned BBPSO variants we introduced only seem to do better when neither algorithm has converged on the mode yet or when the Laplace approximation is poor. So in practice we recommend using standard PSO first.

The AT-BBPSOxp-MC algorithms we introduced still require a larger swarm size or more iterations or both to do as well as standard PSO when standard PSO works well. With the same swarm size and number of iterations AT-BBPSOxp-MC does only slightly worse than standard PSO in terms of maximizing the objective function, but that slight difference often amounts to a large difference in the Metropolis acceptance rate for the resulting MCMC algorithms. When AT-BBPSOxp-MC does do better than standard PSO, the resulting MCMC algorithms do not always have high acceptance rates, at least not without running AT-BBPSOxp-MC with a high swarm size for many iterations. So in practice we recommend using standard PSO first. If the resulting MCMC algorithms have poor acceptance rates or if it seems likely a priori that PSO might have trouble, then it can be useful to explore the

options provided by AT-BBPSOxp-MC with the caveat that a large swarm size and number of iterations may be required.

References

Andrieu, C. and Thoms, J. (2008). "A tutorial on adaptive MCMC." *Statistics and Computing*, 18, 4, 343–373.

Berliner, L. M. (1996). "Hierarchical Bayesian time series models." In *Maximum entropy and Bayesian methods*, 15–22. Springer.

Blum, C. and Li, X. (2008). "Swarm Intelligence in Optimization." In *Swarm Intelligence: Introduction and Applications*, eds. C. Blum and D. Merkle. Springer.

Bradley, J. R., Holan, S. H., and Wikle, C. K. (2015). "Multivariate spatio-temporal models for high-dimensional areal data with application to longitudinal employer-household dynamics." *Annals of Applied Statistics*, 9, 4, 1761–1791.

Bradley, J. R., Wikle, C. K., and Holan, S. H. (2016). "Regionalization of multiscale spatial processes by using a criterion for spatial aggregation error." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*.

Campos, M., Krohling, R. A., and Enriquez, I. (2014). "Bare bones particle swarm optimization with scale matrix adaptation." *Cybernetics, IEEE Transactions on*, 44, 9, 1567–1578.

Carpenter, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M. A., Guo, J., Li, P., and Riddell, A. (2015). "Stan: a probabilistic programming language." *Journal of Statistical Software*.

Chen, Z. and Dunson, D. B. (2003). "Random effects selection in linear mixed models." *Biometrics*, 59, 4, 762–769.

Clerc, M. (2010). *Particle swarm optimization*. John Wiley & Sons.

Clerc, M. and Kennedy, J. (2002). "The particle swarm-explosion, stability, and convergence in a multidimensional complex space." *Evolutionary Computation, IEEE Transactions on*, 6, 1, 58–73.

Eberhart, R. C. and Shi, Y. (2000). "Comparing inertia weights and constriction factors in particle swarm optimization." In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, vol. 1, 84–88. IEEE.

Frühwirth-Schnatter, S. and Tüchler, R. (2008). "Bayesian parsimonious covariance estimation for hierarchical linear mixed models." *Statistics and Computing*, 18, 1, 1–13.

Gelman, A. and Hill, J. (2006). *Data analysis using regression and multilevel/hierarchical models*. Cambridge University Press.

Goldberg, D. E. and Holland, J. H. (1988). "Genetic algorithms and machine learning." *Machine learning*, 3, 2, 95–99.

Hastings, W. K. (1970). "Monte Carlo sampling methods using Markov chains and their applications." *Biometrika*, 57, 1, 97–109.

Hofert, M. (2013). "On Sampling from the Multivariate t Distribution." *The R Journal*, 5, 2, 129–136.

Homan, M. D. and Gelman, A. (2014). "The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo." *The Journal of Machine Learning Research*, 15, 1, 1593–1623.

Hsieh, H.-I. and Lee, T.-S. (2010). "A modified algorithm of bare bones particle swarm optimization." *International Journal of Computer Science Issues*, 7, 11.

Hughes, J. and Haran, M. (2013). "Dimension reduction and alleviation of confounding for spatial generalized linear mixed models." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75, 1, 139–159.

Kennedy, J. (2003). "Bare bones particle swarms." In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, 80–87. IEEE.

Krohling, R., Mendel, E., et al. (2009). "Bare bones particle swarm optimization with Gaussian or Cauchy jumps." In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, 3285–3291. IEEE.

Liu, J. S. (1996). "Metropolized independent sampling with comparisons to rejection sampling and importance sampling." *Statistics and Computing*, 6, 2, 113–119.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). "Equation of state calculations by fast computing machines." *The journal of chemical physics*, 21, 6, 1087–1092.

Neal, R. M. et al. (2011). "MCMC using Hamiltonian dynamics." *Handbook of Markov Chain Monte Carlo*, 2, 113–162.

Porter, A. T., Holan, S. H., and Wikle, C. K. (2015). "Bayesian semiparametric hierarchical empirical likelihood spatial models." *Journal of Statistical Planning and Inference*, 165, 78–90.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Richer, T. J. and Blackwell, T. M. (2006). "The Lévy particle swarm." In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 808–815. IEEE.

Robert, C. and Casella, G. (2013). *Monte Carlo statistical methods*. 2nd ed. Springer Science & Business Media.

Rue, H., Martino, S., and Chopin, N. (2009). "Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations." *Journal of the royal statistical society: Series B (statistical methodology)*, 71, 2, 319–392.

Schervish, M. J. (1997). *Theory of statistics*. Springer Science & Business Media.

Smith, W. and Hocking, R. (1972). "Algorithm AS 53: Wishart variate generator." *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 21, 3, 341–345.

Taylor, B. M. and Diggle, P. J. (2014). "INLA or MCMC? A tutorial and comparative evaluation for spatial prediction in log-Gaussian Cox processes." *Journal of Statistical Computation and Simulation*, 84, 10, 2266–2284.

Tuppadung, Y. and Kurutach, W. (2011). "Comparing nonlinear inertia weights and constriction factors in particle swarm optimization." *International Journal of Knowledge-based and Intelligent Engineering Systems*, 15, 2, 65–70.

Wikle, C. K. et al. (2003). "Hierarchical Models in Environmental Science." *International Statistical Review*, 71, 2, 181–199.

Particle Swarm Optimization Assisted Metropolis Hastings Algorithms

Appendix A: PSO and BBPSO details

[THIS IS JUST PASTED TEXT THAT CAME FROM THE MAIN BODY OF THE DOCUMENT - SECTION NEEDS TO BE WRITTEN IN EARNEST. INCLUDE RING TOPOLOGY STUFF HERE.]

A downside of both versions of BBPSO above is that any particle currently at its group best location does not move due to the definition of the standard deviation term. Several methods have been proposed to overcome this; e.g., see Hsieh and Lee (2010) and Zhang et al. (2011). Zhang et al. (2011) propose using mutation and crossover operations for the group best particle. To do this, the group best particle randomly selects three other distinct particles i_1, i_2, and i_3, and updates:

$$\theta_{ij}(t+1) = p_{i_1 j}(t) + 0.5(p_{i_2 j}(t) - p_{i_3 j}(t)). \tag{1}$$

When combined with BBPSOxp, this becomes

$$\theta_{ij}(t+1) = \begin{cases} p_{i_1 j}(t) + 0.5(p_{i_2 j}(t) - p_{i_3 j}(t)) & \text{with probability } 0.5 \\ g_{ij}(t) & \text{otherwise,} \end{cases} \tag{2}$$

but it also introduces another problem — a particle which moves to its group best location on a single coordinate will have a standard deviation of zero for that coordinate during the next iteration. This is easily overcome by manually setting the standard deviation to a small value (e.g., 0.001), which allows us to define the two base BBPSO algorithms that we will consider: BBPSO-MC and BBPSOxp-MC. BBPSO-MC is standard BBPSO where each particle evolves according to (2) except any current group best particle evolves according to (1). BBPSOxp-MC evolves every particle according to

$$\theta_{ij}(t+1) = \begin{cases} N\left(\frac{p_{ij}(t)+g_{ij}(t)}{2}, \sigma_{ij}^2(t)\right) & \text{with probability } 0.5 \\ g_{ij}(t) & \text{otherwise,} \end{cases} \tag{3}$$

where $\sigma_{ij}(t) = |p_{ij}(t) - g_{ij}(t)|$ if $|p_{ij}(t) - g_{ij}(t)| > 0$ and $\sigma_{ij}(t) = 0.001$ otherwise, except current group best particles evolve according to (2).

Appendix B: Comparing AT-BBPSO, BBPSO, and PSO algorithms

In order to compare AT-BBPSO to other PSO variants, we employ a subset of test functions used in Hsieh and Lee (2010). Each function is listed in Table 1 along with the global maximum and argmax, and the initialization range for the simulations. Further description of many of these functions can be found in Clerc (2010). Each function is randomly initialized in a range that does not contain its global maximum.

Equation	ArgMax	Maximum	Initialization
$Q_1(\boldsymbol{\theta}) = -\sum_{i=1}^{D} \theta_i^2$	$\boldsymbol{\theta}^* = \mathbf{0}$	$Q_1(\boldsymbol{\theta}^*) = 0$	$(50, 100)^D$
$Q_2(\boldsymbol{\theta}) = -\sum_{i=1}^{D} \left(\sum_{j=1}^{i} \theta_j\right)^2$	$\boldsymbol{\theta}^* = \mathbf{0}$	$Q_2(\boldsymbol{\theta}^*) = 0$	$(50, 100)^D$
$Q_3(\boldsymbol{\theta}) = -\sum_{i=1}^{D-1} [100\{\theta_{i+1} + 1 - (\theta_i + 1)^2\} + \theta_i^2]$	$\boldsymbol{\theta}^* = \mathbf{0}$	$Q_3(\boldsymbol{\theta}^*) = 0$	$(15, 30)^D$
$Q_4(\boldsymbol{\theta}) = 9D - \sum_{i=1}^{D} \{\theta_i^2 - \cos(2\pi\theta_i) + 10\}$	$\boldsymbol{\theta}^* = \mathbf{0}$	$Q_4(\boldsymbol{\theta}^*) = 0$	$(2.56, 5.12)^D$
$Q_5(\boldsymbol{\theta}) = -\frac{1}{4000}\|\|\boldsymbol{\theta}\|\|^2 + \prod_{i=1}^{D} \cos\left(\frac{\theta_i}{\sqrt{i}}\right) - 1$	$\boldsymbol{\theta}^* = \mathbf{0}$	$Q_5(\boldsymbol{\theta}^*) = 0$	$(300, 600)^D$
$Q_6(\boldsymbol{\theta}) = 20\exp\left(-0.2\sqrt{\frac{1}{D}\|\|\boldsymbol{\theta}\|\|}\right)$ $+ \exp\left\{\frac{1}{D}\sum_{i=1}^{D} \cos(2\pi\theta_i)\right\} - 20 - \exp(1)$	$\boldsymbol{\theta}^* = \mathbf{0}$	$Q_6(\boldsymbol{\theta}^*) = 0$	$(16, 32)^D$

Table 1: Test functions for evaluating PSO algorithms. The dimension of $\boldsymbol{\theta}$ is D and $||.||$ is the Euclidean norm: $||\boldsymbol{\theta}|| = \sqrt{\sum_{i=1}^{D} \theta_i^2}$.

We use several PSO algorithms in the simulation study. The standard PSO algorithm uses the parameter values suggested by Blum and Li (2008) and Clerc and Kennedy (2002). The AT-BBPSO variants are implemented a wide variety of parameter values, but all have the scale parameter initialized at $\sigma(0) = 1$, and both increment or decrement $\log \sigma$ by $c = 0.1$

every iteration. The AT-PSO variants are initialized at $\omega(0) = 1$ and $\log \omega$ is incremented or decremented by $c = 0.1$ every iteration. In addition, each algorithm is implemented using each of three neighborhood structures. The global neighborhood allows each particle to look at each other particle in the swarm in order to determine its group best — in this case the group best is the swarm best and is used by all particles. The ring-1 neighborhood only allows particles to look at their neighbors as defined by a ring structure. Label each particle with an integer, $0, 1, \ldots, n-1$. Then particle i only looks at particles $i-1$ mod n and $i+1$ mod n in order to determine what its group best is. Addition and subtraction is mod n so that particle 0 looks at particles $n-1$ and 1 to determine its group best. This restricts the flow of information across particles allowing them to explore their nearby space more fully before being pushed in the direction of the rest of the swarm. For less well behaved functions this behavior typically improves the algorithm's ability to find the global max by allowing for more exploration of the objective surface and less exploitation of the best known information. Finally the ring-3 neighborhood is similar to ring-1 but allows each particle to look at its 3 nearest neighbors in each direction, so, for example, particle 1 sees particles 2, 3, 4, n_{swarm}, $n_{swarm} - 1$, and $n_{swarm} - 2$. This still restricts the flow of information across the swarm, but less so than the ring-1 neighborhood.

Each algorithm was used to optimize each objective function for 500 iterations over 100 replications. Initializations were changed across replications but held constant across algorithms. The standard PSO, DI-PSO, and AT-PSO algorithms initialized their velocity terms with $v_{ij}(0) \overset{iid}{\sim} U(-1, 1)$. Tables 2-7 contain the simulation results for objective functions 1-6 respectively (OF1, OF2, etc.). We use several measures to quantify how well each algorithm finds the global maximum. First, each table includes the mean and standard deviation of the absolute difference between the true global maximum and the algorithm's estimated global maximum across all 50 replication. Second, each table includes measures of two convergence criterion — the proportion of the replications that came within 0.01 of the true global

maximum and the proportion that came with 0.0001, denoted by $\widehat{p}_2$ and $\widehat{p}_4$ respectively.

We highlight only some of the features of these tables. First, the BBPSO-MC and AT-BBPSO-MC almost always do worse than their BBPSOxp-MC and AT-BBPSOxp-MC cousins, both in terms of mean absolute difference from the global maximum and in terms of the convergence criterion. The main exception is OF2. Second, for most algorithms the more restrictive neighborhood appears to be result in algorithms which do a better job of finding the global max. This is not universally true, and one interesting class of exceptions are the AT-PSO algorithms. For them, it appears that the target improvement rate (R^*) and the neighborhood interact. When the rate is high a more restrictive neighborhood is preferable, while when the rate is low a less restrictive neighborhood is preferable. Though for OF4 a more restrictive neighborhood always seems preferable. Typically, the AT-PSO algorithms with $R^* = 0.3$ or $R^* = 0.5$ using the ring-1 neighborhood does the best of the AT-PSO algorithms, though there are exceptions.

For the DI-PSO algorithms often there is a parameter-neighborhood combination that does well, typically from setting $\alpha = 200$ (20% of the 500 iterations) and $\beta = 1$ and using either the ring-1 or ring-3 neighborhood. Call this combination with the ring-1 neighborhood the default DI-PSO algorithm. The default combination typically does the best of all the DI-PSO algorithms but they can sometimes still do much worse than the best alternatives (e.g., for OF2). In the AT-BBPSO algorithms, lower values of *df* often result in better algorithms, though sometimes the difference is small. The impact of R^* is much smaller, though $R^* = 0.3$ or $R^* = 0.5$ seem to be the safest choices, though other classes of algorithms will often perform better.

In general, the standard PSO algorithm and the BBPSOxp-MC algorithm do pretty well and should serve as baselines, typically using the ring-1 neighborhood. For OF1, PSO meets both convergence criterion 100% of the time in our simulations, while only BBPSOxp-MC has $\widehat{p}_2 = 1$. None of the AT-BBPSOxp-MC algorithms are able to hit the second criterion

at a high rate, though our recommended DI-PSO and AT-PSO algorithms are able to. For OF2, the standard PSO algorithm using the ring-3 neighborhood is the best performing of the baseline algorithms, with $\widehat{p}_2 = 0.86$ and $\widehat{p}_4 = 0.1$. The best performing algorithm overall is AT-PSO with $R^* = 0.3$, yielding $\widehat{p}_2 = \widehat{p}_4 = 1$, and AT-PSO with $R^* = 0.5$ is in a close second with $\widehat{p}_2 = 1$ and $\widehat{p}_4 = 0.72$. None of the AT-BBPSO or DI-PSO algorithms even come close to the baseline. OF3 is much more challenging and almost no algorithm has nonzero convergence rates. The best baseline algorithm is BBPSOxp-MC with the ring-1 neighborhood, with a mean absolute difference from the true max of 18.77 respectively. The ring-1 PSO algorithm has a mean of 25.95, which we will use as another baseline for this objective function. Many of the ring-1 AT-BBPSOxp-MC algorithms are competitive with the baselines, but essentially nothing else is in terms of mean absolute difference. However, the ring-1 AT-PSO algorithm with $R^* = 0.5$ does meet both convergence criterion in 2% of replication, while no other algorithm ever meets the criterion.

The best baseline algorithm for OF4 may be ring-1 PSO, with a mean absolute difference of 0.13 and convergence proportions of 0.90 and 0.86. However, the BBPSOxp-MC algorithm is comparable with a mean of 0.01 and $\widehat{p}_2 = 0.86$, but $\widehat{p}_4 = 0$. Which is better depends on how precise you need to estimate the global maximum. All of the AT-BBPSOxp-MC algorithms are similar to BBPSOxp-MC with a near zero mean, $\widehat{p}_2$ approaching 1 and $\widehat{p}_4$ essentially 0. None of the DI-PSO algorithms are competitive and even the best AT-PSO algorithms are worse than the baseline PSO algorithm. OF5 is another difficult one where most algorithms never meet any of the convergence criterion. ring-1 PSO and ring-3 PSO are the best baselines with mean absolute differences of 0.06 and 0.07 respectively. With the ring-1 or ring-3 neighborhood, AT-BBPSOxp-MC algorithms with $df = 1$ are comparable to both baselines, The best performing algorithms in terms of mean absolute difference are the ring-1 AT-PSO algorithms with $R^* = 0.3$ or 0.5. OF6 is the most difficult with many local optima. The baseline algorithms do poorly, though ring-3 PSO and ring-1 PSO have high

convergence proportions despite high mean absolute differences. The AT-BBPSOxp-MC algorithms with $df = 1$ and $R^* = 0.3$ or 0.5 do much better in terms of the mean absolute difference, but almost never meet the convergence criterion. The ring-3 DI-PSO algorithm with $\alpha = 200$ and $\beta = 1$ is the best converging algorithm, with $\widehat{p}_2 = 0.7$ and $\widehat{p}_4 = 0.68$, though ring-3 AT-PSO with $R^* = 0.1$ has $\widehat{p}_2 = 0.7$ and $\widehat{p}_4 = 0.12$

In general ring-1 or ring-3 PSO is a great baseline algorithm that works reasonably well over a wide variety of circumstances. The AT-BBPSOxp-MC algorithms with $df = 1$ and $R^* = 0.3$ or 0.5 also tend to do reasonably well, though their strength seems to be in finding the region around the global max quickly. When the optimization problem is not too difficult, the best AT-BBPSOxp-MC algorithms have trouble converging as fast as the best alternatives, but when the optimization problem is difficult enough that convergence is unlikely in the number of allotted iterations AT-BBPSOxp-MC often gets closer to the global maximum than the alternatives on average.

The DI-PSO and AT-PSO algorithms similar conceptually, but often yield very different results. DI-PSO deterministically reduces the inertia parameter over time in the same manner given a fixed set of parameter values (α and β), while AT-PSO dynamically adjusts the inertia parameter to hit a target improvement rate. Figure 1 plots the inertia over time for the DI-PSO algorithm with $\alpha = 200$, i.e. 20% of the total number of PSO iterations, and $\beta = 1$, and observed inertia over time for one replication of the $AT - PSO$ algorithm with target rate $R^* = 0.5$ and ring-1 neighborhood for OF1 and one replication for OF8. All three algorithms have an initial inertia of $\omega(0) = 1$. While DI-PSO smoothly decreases its inertia with a slowly decreasing rate, for OF1 AT-PSO very quickly drops its inertia to about 0.5 then bounces around around that point. It also jumps up above 1 initially, imploring the particles to cast a wider net in search of higher value areas of the search space. This is pretty typical behavior for the inertia parameter of AT-PSO — it tends to bounce around a level which is approximately the average over time of the DI-PSO's inertia, though lower values of

R^* will result in higher inertias. In this way, AT-PSO alternates periods of search (relatively high inertia) and periods of exploitation (relatively low inertia). The main exception to this pattern is when AT-PSO converges around a local maximum. In this case, inertia plummets to zero as the particles settle down. This is precisely what happens for OF6 in Figure 1, though in this case the maximum is not global — Table 7 indicates that ring-1 AT-PSO with $R^* = 0.5$ very rarely converged to the global max. In optimization problems with multiple local optima, both AT-PSO and AT-BBPSO variants can exhibit this behavior prematurely converge to a local optima, so they may not be advantageous for those problems. Reducing the target improvement rate (R^*) can ameliorate this problem though. We used $c = 0.1$ for all AT type algorithms in this section, but decreasing c will slow down how fast the inertia parameter is adapted in AT-PSO algorithms and allow for a longer high inertia exploration phase, mimicing the DI-PSO algorithms to some extent. This may also help in problems with many local optima.

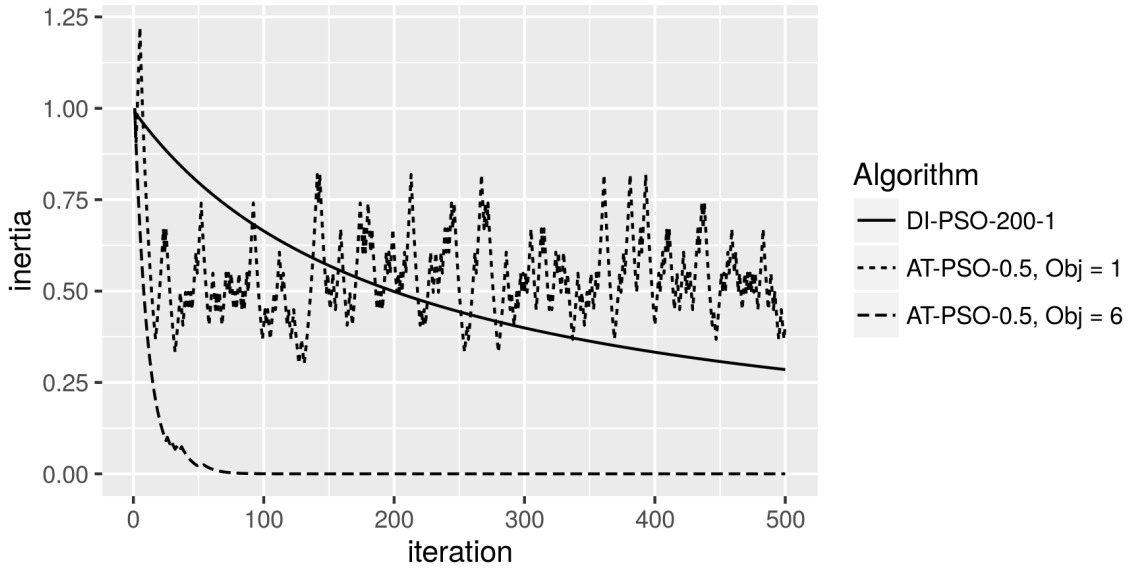

Figure 1: Inertia over time for the DI-PSO algorithm with $\alpha = 200$ and $\beta = 1$, and for one replication of the AT-PSO-0.5 algorithm for each of OFs 1 and 6.

Based on these simulations, our default recommendation is conditional. In problems where convergence is important, such as when finding the posterior mode to use for the normal approximation in a Metropolis proposal, we recommend AT-PSO with $R^* = 0.3$ or 0.5 and using a fairly restricted neighborhood (e.g., ring-1 or ring-3). These algorithms are often the best at meeting convergence criteria, especially in problems with few or no extra local optima. In difficult problems where convergence is less important (e.g., in machine learning) an AT-BBPSO variant with $df = 1$ and $R^* = 0.3$ or 0.5, and again with a restrictive neighborhood seems appropriate. These sorts of algorithms seem to do a better job of getting into a region around the global max, though not as well at searching through that region. This bipartite strategy suggests a combined strategy: use AT-BBPSO or some algorithm at first in order to quickly find a good region of the search space, then use AT-PSO to quickly search through that region.

A Deriving the Hessian

The log posterior in the fully parameterized Poisson case can be written as

$$\begin{aligned}
\log p(\boldsymbol{\beta}, \boldsymbol{L}, \boldsymbol{\delta} | \boldsymbol{z}, \boldsymbol{X}, \boldsymbol{S}) &= \text{constant} + \sum_{i=1}^{n} (y_i z_i - \exp(y_i)) + \sum_{k=1}^{r} \frac{d-k+1}{2} \log \ell_{kk}^2 \\
&\quad - \frac{1}{2} \left\{ \boldsymbol{\delta}' \boldsymbol{L} \boldsymbol{L}' \boldsymbol{\delta} + \frac{(\boldsymbol{\beta} - \boldsymbol{b})'(\boldsymbol{\beta} - \boldsymbol{b})}{v^2} + \operatorname{tr}(\boldsymbol{E} \boldsymbol{L} \boldsymbol{L}') \right\} \\
&= \text{constant} + \sum_{i=1}^{n} \{ z_i y_i - \exp(y_i) \} + \frac{1}{2} \boldsymbol{R}_r \operatorname{vech}(\log \boldsymbol{L}^2) \\
&\quad - \frac{1}{2} \left[\operatorname{vech}(\boldsymbol{L})' \boldsymbol{M}_r \left\{ \boldsymbol{K}_r' (\boldsymbol{\delta}\boldsymbol{\delta}' \otimes \boldsymbol{I}_r) \boldsymbol{K}_r + (\boldsymbol{I}_r \otimes \boldsymbol{E}) \right\} \boldsymbol{M}_r' \operatorname{vech}(\boldsymbol{L}) + \frac{(\boldsymbol{\beta} - \boldsymbol{b})'(\boldsymbol{\beta} - \boldsymbol{b})}{v^2} \right]
\end{aligned}$$

where $y_i = \boldsymbol{x}_i' \boldsymbol{\beta} + \boldsymbol{s}_i' \boldsymbol{\delta}$, $\otimes$ is the Kronecker product, $\boldsymbol{I}_r$ is the $r \times r$ identity matrix, and $\boldsymbol{R}_r$ is an $r(r+1)/2 \times r(r+1)/2$ matrix of zeroes except the $(r+1)k - k(k+1)/2 + k - r$st diagonal element is equal to $d - k + 1$ for $k = 1, 2, \ldots, r$, corresponding to locations in $\operatorname{vech}(\boldsymbol{L})$ that store the diagonal elements of $\boldsymbol{L}$. In $\operatorname{vech}(\log \boldsymbol{L}^2)$ the log and power are applied element-

wise. The expansions of $\boldsymbol{\delta}'\boldsymbol{L}\boldsymbol{L}'\boldsymbol{\delta}$ and $\operatorname{tr}(\boldsymbol{E}\boldsymbol{L}\boldsymbol{L}')$ can be derived from the properties of the trace operator, Kronecker products, and the following identities.

$$\begin{aligned}\boldsymbol{\delta}'\boldsymbol{L}\boldsymbol{L}'\boldsymbol{\delta} &= \operatorname{vec}(\boldsymbol{L}'\boldsymbol{\delta})'\operatorname{vec}(\boldsymbol{L}'\boldsymbol{\delta})\\ \operatorname{vec}(\boldsymbol{L}'\boldsymbol{\delta}) &= (\boldsymbol{\delta}'\otimes\boldsymbol{I}_r)\operatorname{vec}(\boldsymbol{L}')\\ \operatorname{vec}(\boldsymbol{L}') &= \boldsymbol{K}_r\operatorname{vec}(\boldsymbol{L}) = \boldsymbol{K}_r\boldsymbol{M}_r'\operatorname{vech}(\boldsymbol{L})\end{aligned}$$

and assuming $\boldsymbol{E} = \boldsymbol{C}\boldsymbol{C}'$, then $\operatorname{tr}(\boldsymbol{L}'\boldsymbol{C}\boldsymbol{C}'\boldsymbol{L}) = \operatorname{vec}(\boldsymbol{C}'\boldsymbol{L})'\operatorname{vec}(\boldsymbol{C}'\boldsymbol{L})$ where $\operatorname{vec}(\cdot)$ is the vectorization operation which stacks each column of its argument on top of each other into a single column vector, $\operatorname{vech}(\cdot)$ is the half-vectorization operator which is similar but omits elements above the diagonal, tr is the trace operator, $\boldsymbol{K}_r$ is the $r^2 \times r^2$ commutation matrix such that for any $r \times r$ matrix $\boldsymbol{L}$, $\operatorname{vec}(\boldsymbol{L}') = \boldsymbol{K}_r\operatorname{vec}(\boldsymbol{L})$, $\boldsymbol{M}_r$ is the $r^2 \times r(r+1)/2$ elimination matrix such that for $\operatorname{vech}(\boldsymbol{L}) = \boldsymbol{M}_r\operatorname{vec}(\boldsymbol{L})$ and if additionally $\boldsymbol{L}$ is lower triangular, $\operatorname{vec}(\boldsymbol{L}) = \boldsymbol{M}_r'\operatorname{vech}(\boldsymbol{L})$ (Magnus and Neudecker, 1980; Magnus, 1988). Let $\boldsymbol{E}_{ij}$ be an $r \times r$ matrix of zeroes with a one in only its (i,j)th position, $\boldsymbol{u}_{ij} = \operatorname{vech}(\boldsymbol{E}_{ij})$, and $\boldsymbol{e}_i$ be a r-dimensional column vector of zeroes with a one in the ith row. Then $\boldsymbol{K}_r$ and $\boldsymbol{M}_r$ can be written as

$$\boldsymbol{K}_r = \sum_{i=1}^{r}\sum_{j=1}^{r}\boldsymbol{E}_{ij}\otimes\boldsymbol{E}_{ij}' \qquad \text{and} \qquad \boldsymbol{M}_r = \sum_{i\geq j}^{r}\boldsymbol{u}_{ij}\otimes\boldsymbol{e}_j'\otimes\boldsymbol{e}_i'.$$

Now the first derivatives of the log posterior are given by

$$\begin{aligned}\frac{\partial\log p}{\partial\boldsymbol{\beta}} &= \sum_{i=1}^{n}(z_i - e^{y_i})\boldsymbol{x}_i' - \frac{(\boldsymbol{\beta}-\boldsymbol{b})'}{v^2},\\ \frac{\partial\log p}{\partial\boldsymbol{\delta}} &= \sum_{i=1}^{n}(z_i - e^{y_i})\boldsymbol{s}_i' - \boldsymbol{\delta}'\boldsymbol{L}\boldsymbol{L}',\\ \frac{\partial\log p}{\partial\operatorname{vech}(\boldsymbol{L})} &= -\operatorname{vech}(\boldsymbol{L})'\boldsymbol{M}_r\left[\boldsymbol{K}_r'(\boldsymbol{\delta}\boldsymbol{\delta}'\otimes\boldsymbol{I}_r)\boldsymbol{K}_r + (\boldsymbol{I}_r\otimes\boldsymbol{E})\right]\boldsymbol{M}_r' + \boldsymbol{R}_r\operatorname{vech}(1/\boldsymbol{L}),\end{aligned}$$

where in $\text{vech}(1/\boldsymbol{L})$ the division is applied element-wise. Next the second derivatives are

$$\begin{aligned}
\frac{\partial^2 \log p}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}'} &= -\sum_{i=1}^{n} e^{y_i} \boldsymbol{x}_i \boldsymbol{x}_i' - \frac{1}{v^2} \boldsymbol{I}_p \\
\frac{\partial^2 \log p}{\partial \boldsymbol{\delta} \partial \boldsymbol{\delta}'} &= -\sum_{i=1}^{n} e^{y_i} \boldsymbol{s}_i \boldsymbol{s}_i' - \boldsymbol{L}\boldsymbol{L}' \\
\frac{\partial^2 \log p}{\partial \,\text{vech}(\boldsymbol{L}) \partial \,\text{vech}(\boldsymbol{L})'} &= -\boldsymbol{M}_r \left[\boldsymbol{K}_r'(\boldsymbol{\delta}\boldsymbol{\delta}' \otimes \boldsymbol{I}_r)\boldsymbol{K}_r + (\boldsymbol{I}_r \otimes \boldsymbol{E})\right] \boldsymbol{M}_r' - \boldsymbol{R}_r \,\text{diag}\left(\text{vech}(\boldsymbol{L})^{-2}\right) \\
\frac{\partial^2 \log p}{\partial \boldsymbol{\beta} \partial \boldsymbol{\delta}'} &= -\sum_{i=1}^{n} e^{y_i} \boldsymbol{s}_i \boldsymbol{x}_i' \\
\frac{\partial^2 \log p}{\partial \boldsymbol{\beta} \partial \,\text{vech}(\boldsymbol{L})'} &= \boldsymbol{0}_{p \times r(r+1)/2} \\
\frac{\partial^2 \log p}{\partial \boldsymbol{\delta} \partial \,\text{vech}(\boldsymbol{L})'} &= -\frac{\partial \boldsymbol{\delta}' \boldsymbol{L}\boldsymbol{L}'}{\partial \,\text{vech}(\boldsymbol{L})'} = -(\boldsymbol{\delta}' \otimes \boldsymbol{I}_r)(\boldsymbol{I}_{r^2} + \boldsymbol{K}_r)(\boldsymbol{L} \otimes \boldsymbol{I}_r)\boldsymbol{M}_r',
\end{aligned}$$

where $\text{diag}(\text{vech}(\boldsymbol{L})^{-2})$ is a diagonal matrix with the elements of $\text{vech}(\boldsymbol{L})$ raised to the power -2 along the diagonal. The last second derivative matrix can be derived using repeated application of the chain rule and from the following facts for any $r \times r$ matrix $\boldsymbol{A}$ (Magnus and Neudecker, 2005):

$$\begin{aligned}
\frac{\partial \,\text{vec}(\boldsymbol{A}\boldsymbol{A}')}{\partial \,\text{vec}(\boldsymbol{A})} &= (\boldsymbol{I}_{r^2} + \boldsymbol{K}_r)(\boldsymbol{A} \otimes \boldsymbol{I}_r) \\
\frac{\partial \boldsymbol{A}\boldsymbol{\delta}}{\partial \,\text{vec}(\boldsymbol{A})} &= \boldsymbol{\delta}' \otimes \boldsymbol{I}_r.
\end{aligned}$$

Then, using the chain rule for lower triangular $\boldsymbol{L}$ we have

$$\begin{aligned}
\frac{\partial \boldsymbol{\delta}' \boldsymbol{L}\boldsymbol{L}'}{\partial \,\text{vech}(\boldsymbol{L})'} &= \frac{\partial \boldsymbol{\delta}' \boldsymbol{L}\boldsymbol{L}'}{\partial \boldsymbol{L}\boldsymbol{L}'\boldsymbol{\delta}} \frac{\partial \boldsymbol{L}\boldsymbol{L}'\boldsymbol{\delta}}{\partial \,\text{vec}(\boldsymbol{L}\boldsymbol{L}')} \frac{\partial \,\text{vec}(\boldsymbol{L}\boldsymbol{L}')}{\partial \,\text{vec}(\boldsymbol{L})} \frac{\partial \,\text{vec}(\boldsymbol{L})}{\partial \,\text{vech}(\boldsymbol{L})} \frac{\partial \,\text{vech}(\boldsymbol{L})}{\partial \,\text{vech}(\boldsymbol{L})'} \\
&= \boldsymbol{I}_r \frac{\partial \boldsymbol{L}\boldsymbol{L}'\boldsymbol{\delta}}{\partial \,\text{vec}(\boldsymbol{L}\boldsymbol{L}')} \frac{\partial \,\text{vec}(\boldsymbol{L}\boldsymbol{L}')}{\partial \,\text{vec}(\boldsymbol{L})} \boldsymbol{M}_r' \boldsymbol{I}_{r(r+1)/2} \\
&= (\boldsymbol{\delta}' \otimes \boldsymbol{I}_r)(\boldsymbol{I}_{r^2} + \boldsymbol{K}_r)(\boldsymbol{L} \otimes \boldsymbol{I}_r)\boldsymbol{M}_r'.
\end{aligned}$$

Finally, the Hessian is

$$\boldsymbol{H} = \begin{bmatrix}
\frac{\partial^2 \log p}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}'} & \frac{\partial^2 \log p}{\partial \boldsymbol{\beta} \partial \boldsymbol{\delta}'} & \frac{\partial^2 \log p}{\partial \boldsymbol{\beta} \partial \,\text{vech}(\boldsymbol{L})'} \\
\frac{\partial^2 \log p}{\partial \boldsymbol{\delta} \partial \boldsymbol{\beta}'} & \frac{\partial^2 \log p}{\partial \boldsymbol{\delta} \partial \boldsymbol{\delta}'} & \frac{\partial^2 \log p}{\partial \boldsymbol{\delta} \partial \,\text{vech}(\boldsymbol{L})'} \\
\frac{\partial^2 \log p}{\partial \,\text{vech}(\boldsymbol{L}) \partial \boldsymbol{\beta}'} & \frac{\partial^2 \log p}{\partial \,\text{vech}(\boldsymbol{L}) \partial \boldsymbol{\delta}'} & \frac{\partial^2 \log p}{\partial \,\text{vech}(\boldsymbol{L}) \partial \,\text{vech}(\boldsymbol{L})'}
\end{bmatrix}.$$

A Alternative MCMC algorithms

[CURRENTLY THIS SECTION ONLY HAS GENERIC VERSIONS OF THE ALGORTIHMS. DETAILS FOR OUR CASES?] This section describes the alternative MCMC algorithms used for comparisons with PSO assisted Metropolis-Hastings algorithms. In particular, we use two types of adaptive random walk Metropolis within Gibbs algorithms. In the generic problem, suppose we wish to sample from the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y})$ using MCMC methods. Suppose that $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ and that the full conditional $p(\boldsymbol{\theta}_1|\boldsymbol{\theta}_2, \boldsymbol{y})$ can be sampled from easily while the full conditional for $\boldsymbol{\theta}_2$ may be intractable. Then a single move random walk Metropolis within Gibbs (RWwG) algorithm for this problem is as follows.

Algorithm 1 (Single move random walk Metropolis within Gibbs) *Given target posterior $p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2|\boldsymbol{y})$ where $\boldsymbol{\theta}_2 = (\theta_{21}, \theta_{22}, \ldots, \theta_{2n})$, it is easy to sample from $p(\boldsymbol{\theta}_1|\boldsymbol{\theta}_2, \boldsymbol{y})$, and given that the support of $p(\boldsymbol{\theta}_2|\boldsymbol{\theta}_1, \boldsymbol{y})$ is $\Re^n$, iteration $t+1$ is obtained from iteration t via*

1. *Draw $\boldsymbol{\theta}_1^{(t+1)} \sim p(\boldsymbol{\theta}_1|\boldsymbol{\theta}_2^{(t)}, \boldsymbol{y})$*

2. *For $k = 1, 2 \ldots, n$*

   Draw $\theta_{2k}^{(prop)} \sim N(\theta_{2k}^{(t)}, \eta_k)$ and form the Metropolis acceptance ratio

$$a_k^{(t)} = \frac{p(\theta_{2k}^{(prop)}|\boldsymbol{\theta}_1^{(t+1)}, \theta_{21}^{(t+1)}, , \ldots, \theta_{2k-1}^{(t+1)}, \theta_{2k+1}^{(t)}, \ldots, \theta_{2n}^{(t)}, \boldsymbol{y})}{p(\theta_{2k}^{(t)}|\boldsymbol{\theta}_1^{(t+1)}, \theta_{21}^{(t+1)}, , \ldots, \theta_{2k-1}^{(t+1)}, \theta_{2k+1}^{(t)}, \ldots, \theta_{2n}^{(t)}, \boldsymbol{y})}.$$

   Then set $\theta_{2k}^{(t+1)} = \theta_{2k}^{(prop)}$ with probability $r_k^{(t)} = \min(a_k, 1)$ and otherwise set $\theta_{2k}^{(t+1)} = \theta_{2k}^{(t)}$

A major problem with this algorithm is that the η_ks must be selected so that the algorithm Metropolis steps accept a reasonable amount of time. According to Gelman et al. (1996) the optimal acceptance rate in a narrow set of problems is 0.44 for a single dimensional random walk, though this is often used as a guideline for more complex problems. We

adaptively tune η_k during the burn-in period of the chain in a manner discussed in Andrieu and Thoms (2008). The computed Metropolis acceptance probability for θ_{2k} is denoted by $r_k^{(t)} = \min(a_k^{(t)}, 1)$; let r^* denote the target acceptance probability. Then we evolve η_k over time via

$$\log\ \eta_k^{(t+1)} = \log\ \eta_k^{(t)} + \gamma^{(t+1)}(r_k^{(t)} - r^*)$$

where $\gamma^{(t)}$ is a fixed sequence decreasing to zero fast enough to enough that the algorithm converges. The intuition here is that if the acceptance rate is too high, we can increase the effective search space by increasing the random walk standard deviation, while if it is too low, we can decrease the effective search space by decreasing the random walk standard deviation. We use $r^* = 0.44$. A tuning method such as this implies that $(\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \dots)$ is no longer a Markov chain, and additional conditions are required to ensure convergence to the target posterior distribution. So in practice our RWwG algorithms run Algorithm 1 with the tuning method described above until convergence and until the η_ks settle into a rough equilibrium, then we use Algorithm 1 without tuning but using the last known values of the η_ks. In practice this is equivalent setting $\gamma^{(t)} = 0$ after the burn-in period. During the burn-in we set $\gamma^{(t)} = 1$ and we initialize at $\eta_k^{(0)} = 1$ for all k.

An alternative to RWwG algorithms are block random walk Metropolis within Gibbs algorithms (B-RWwG). Suppose we have an estimate of the covariance structure between the elements of $\boldsymbol{\theta}_2$ in the posterior. Then

Algorithm 2 (Block random walk Metropolis within Gibbs) *Given target posterior* $p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2|\boldsymbol{y})$ *where it is easy to sample from* $p(\boldsymbol{\theta}_1|\boldsymbol{\theta}_2, \boldsymbol{y})$*, the support of* $p(\boldsymbol{\theta}_2|\boldsymbol{\theta}_1, \boldsymbol{y})$ *is* $\Re^n$*, and given an estimate* $\boldsymbol{\Sigma}$ *of* $\mathrm{cov}(\boldsymbol{\theta}_2|\boldsymbol{y})$*, iteration* $t+1$ *is obtained from iteration* t *via*

1. *Draw* $\boldsymbol{\theta}_1^{(t+1)} \sim p(\boldsymbol{\theta}_1|\boldsymbol{\theta}_2^{(t)}, \boldsymbol{y})$

2. *Draw* $\boldsymbol{\theta}_2^{(prop)} \sim N(\boldsymbol{\theta}_2^{(t)}, \eta\boldsymbol{\Sigma})$ *and form the Metropolis acceptance ratio*

$$a = \frac{p(\boldsymbol{\theta}_2^{(prop)}|\boldsymbol{\theta}_1^{(t+1)}, \boldsymbol{y})}{p(\boldsymbol{\theta}_2^{(t)}|\boldsymbol{\theta}_1^{(t+1)}, \boldsymbol{y})}.$$

Then set $\boldsymbol{\theta}_2^{(t+1)} = \boldsymbol{\theta}_2^{(prop)}$ *with probability* $\min(a, 1)$ *and otherwise set* $\boldsymbol{\theta}_2^{(t+1)} = \boldsymbol{\theta}_2^{(t)}$.

We make this algorithm adaptive as well by tuning η in the same fashion as the η_ks above. The main difference is that this algorithm must be initialized with a crude estimate of $\boldsymbol{\Sigma}$ by running, e.g., Algorithm 1.

Obj = 1	Global nbhd				Ring-3 nbhd				Ring-1 nbhd			
Algorithm	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$
PSO	0.00	0.01	0.98	0.92	0.00	0.00	1.00	1.00	0.00	0.00	1.00	1.00
BBPSO-MC	0.12	0.04	0.00	0.00	0.09	0.03	0.00	0.00	0.04	0.02	0.06	0.00
BBPSOxp-MC	0.02	0.01	0.06	0.00	0.01	0.01	0.60	0.00	0.00	0.00	1.00	0.00
AT-BBPSO-MC												
$df = 1,\ R^* = 0.1$	0.02	0.01	0.02	0.00	0.01	0.01	0.22	0.00	0.00	0.00	0.96	0.00
$df = 1,\ R^* = 0.3$	0.02	0.01	0.06	0.00	0.01	0.00	0.22	0.00	0.00	0.00	0.96	0.00
$df = 1,\ R^* = 0.5$	0.02	0.01	0.04	0.00	0.01	0.00	0.10	0.00	0.00	0.00	0.96	0.00
$df = 1,\ R^* = 0.7$	0.02	0.01	0.06	0.00	0.01	0.01	0.22	0.00	0.00	0.00	0.98	0.00
$df = 3,\ R^* = 0.1$	0.04	0.01	0.02	0.00	0.03	0.01	0.00	0.00	0.01	0.00	0.70	0.00
$df = 3,\ R^* = 0.3$	0.04	0.02	0.00	0.00	0.03	0.01	0.00	0.00	0.01	0.00	0.56	0.00
$df = 3,\ R^* = 0.5$	0.04	0.01	0.00	0.00	0.03	0.01	0.02	0.00	0.01	0.00	0.60	0.00
$df = 3,\ R^* = 0.7$	0.04	0.01	0.00	0.00	0.03	0.01	0.02	0.00	0.01	0.00	0.64	0.00
$df = 5,\ R^* = 0.1$	0.06	0.02	0.00	0.00	0.04	0.02	0.00	0.00	0.01	0.01	0.32	0.00
$df = 5,\ R^* = 0.3$	0.06	0.02	0.00	0.00	0.04	0.01	0.00	0.00	0.02	0.01	0.26	0.00
$df = 5,\ R^* = 0.5$	0.06	0.02	0.00	0.00	0.04	0.01	0.00	0.00	0.01	0.01	0.34	0.00
$df = 5,\ R^* = 0.7$	0.06	0.02	0.00	0.00	0.04	0.01	0.00	0.00	0.01	0.01	0.36	0.00
$df = \infty,\ R^* = 0.1$	0.13	0.04	0.00	0.00	0.09	0.04	0.00	0.00	0.04	0.02	0.00	0.00
$df = \infty,\ R^* = 0.3$	0.12	0.04	0.00	0.00	0.09	0.02	0.00	0.00	0.04	0.02	0.02	0.00
$df = \infty,\ R^* = 0.5$	0.13	0.04	0.00	0.00	0.10	0.03	0.00	0.00	0.04	0.02	0.02	0.00
$df = \infty,\ R^* = 0.7$	0.12	0.04	0.00	0.00	0.09	0.03	0.00	0.00	0.03	0.02	0.00	0.00
AT-BBPSOxp-MC												
$df = 1,\ R^* = 0.1$	0.00	0.00	0.98	0.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.20
$df = 1,\ R^* = 0.3$	0.00	0.00	0.98	0.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.32
$df = 1,\ R^* = 0.5$	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.26
$df = 1,\ R^* = 0.7$	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.20
$df = 3,\ R^* = 0.1$	0.01	0.00	0.90	0.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.10
$df = 3,\ R^* = 0.3$	0.01	0.00	0.94	0.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.10
$df = 3,\ R^* = 0.5$	0.01	0.00	0.88	0.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.06
$df = 3,\ R^* = 0.7$	0.01	0.00	0.90	0.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.08
$df = 5,\ R^* = 0.1$	0.01	0.00	0.64	0.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.08
$df = 5,\ R^* = 0.3$	0.01	0.00	0.50	0.00	0.00	0.00	0.96	0.00	0.00	0.00	1.00	0.02
$df = 5,\ R^* = 0.5$	0.01	0.00	0.58	0.00	0.00	0.00	0.98	0.00	0.00	0.00	1.00	0.02
$df = 5,\ R^* = 0.7$	0.01	0.00	0.54	0.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.04
$df = \infty,\ R^* = 0.1$	0.02	0.01	0.00	0.00	0.01	0.00	0.58	0.00	0.00	0.00	1.00	0.00
$df = \infty,\ R^* = 0.3$	0.02	0.01	0.06	0.00	0.01	0.00	0.52	0.00	0.00	0.00	1.00	0.02
$df = \infty,\ R^* = 0.5$	0.02	0.01	0.10	0.00	0.01	0.01	0.60	0.00	0.00	0.00	1.00	0.00
$df = \infty,\ R^* = 0.7$	0.02	0.01	0.10	0.00	0.01	0.00	0.58	0.00	0.00	0.00	1.00	0.00
DI-PSO												

Obj = 2	Global nbhd				Ring-3 nbhd				Ring-1 nbhd			
Algorithm	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$
PSO	15.69	80.28	0.66	0.42	0.01	0.01	0.86	0.10	1.92	4.16	0.00	0.00
BBPSO-MC	0.43	0.13	0.00	0.00	0.47	0.23	0.00	0.00	0.72	0.53	0.00	0.00
BBPSOxp-MC	2.10	1.59	0.00	0.00	3.64	4.33	0.00	0.00	3.20	7.55	0.00	0.00
AT-BBPSO-MC												
$df = 1,\ R^* = 0.1$	0.17	0.10	0.00	0.00	0.18	0.09	0.00	0.00	0.10	0.08	0.00	0.00
$df = 1,\ R^* = 0.3$	0.19	0.08	0.00	0.00	0.17	0.09	0.00	0.00	0.10	0.06	0.00	0.00
$df = 1,\ R^* = 0.5$	0.17	0.09	0.00	0.00	0.19	0.11	0.00	0.00	0.16	0.14	0.00	0.00
$df = 1,\ R^* = 0.7$	0.18	0.09	0.00	0.00	0.16	0.09	0.00	0.00	0.11	0.09	0.00	0.00
$df = 3,\ R^* = 0.1$	0.20	0.08	0.00	0.00	0.18	0.10	0.00	0.00	0.17	0.17	0.00	0.00
$df = 3,\ R^* = 0.3$	0.20	0.08	0.00	0.00	0.18	0.08	0.00	0.00	0.12	0.08	0.00	0.00
$df = 3,\ R^* = 0.5$	0.20	0.09	0.00	0.00	0.19	0.09	0.00	0.00	0.12	0.07	0.00	0.00
$df = 3,\ R^* = 0.7$	0.21	0.08	0.00	0.00	0.22	0.08	0.00	0.00	0.12	0.10	0.00	0.00
$df = 5,\ R^* = 0.1$	0.26	0.10	0.00	0.00	0.22	0.10	0.00	0.00	0.16	0.09	0.00	0.00
$df = 5,\ R^* = 0.3$	0.27	0.13	0.00	0.00	0.24	0.09	0.00	0.00	0.17	0.11	0.00	0.00
$df = 5,\ R^* = 0.5$	0.25	0.08	0.00	0.00	0.24	0.10	0.00	0.00	0.16	0.10	0.00	0.00
$df = 5,\ R^* = 0.7$	0.27	0.09	0.00	0.00	0.20	0.09	0.00	0.00	0.19	0.10	0.00	0.00
$df = \infty,\ R^* = 0.1$	0.45	0.18	0.00	0.00	0.47	0.17	0.00	0.00	0.58	0.65	0.00	0.00
$df = \infty,\ R^* = 0.3$	0.52	0.22	0.00	0.00	0.46	0.20	0.00	0.00	0.57	0.38	0.00	0.00
$df = \infty,\ R^* = 0.5$	0.42	0.17	0.00	0.00	0.42	0.15	0.00	0.00	0.66	0.63	0.00	0.00
$df = \infty,\ R^* = 0.7$	0.45	0.20	0.00	0.00	0.46	0.18	0.00	0.00	0.60	0.46	0.00	0.00
AT-BBPSOxp-MC												
$df = 1,\ R^* = 0.1$	1.32	0.93	0.00	0.00	0.97	0.95	0.00	0.00	0.29	0.36	0.00	0.00
$df = 1,\ R^* = 0.3$	1.33	1.25	0.00	0.00	0.98	0.86	0.00	0.00	0.23	0.26	0.00	0.00
$df = 1,\ R^* = 0.5$	1.55	1.65	0.00	0.00	1.09	1.76	0.00	0.00	0.27	0.44	0.00	0.00
$df = 1,\ R^* = 0.7$	1.07	0.83	0.00	0.00	1.10	1.05	0.00	0.00	0.19	0.11	0.00	0.00
$df = 3,\ R^* = 0.1$	0.72	0.56	0.00	0.00	0.69	0.50	0.00	0.00	0.29	0.24	0.00	0.00
$df = 3,\ R^* = 0.3$	0.70	0.45	0.00	0.00	0.95	1.25	0.00	0.00	0.25	0.31	0.00	0.00
$df = 3,\ R^* = 0.5$	0.63	0.35	0.00	0.00	0.77	0.66	0.00	0.00	0.36	0.48	0.00	0.00
$df = 3,\ R^* = 0.7$	0.64	0.49	0.00	0.00	0.73	0.47	0.00	0.00	0.32	0.42	0.00	0.00
$df = 5,\ R^* = 0.1$	0.69	0.42	0.00	0.00	1.83	5.74	0.00	0.00	0.77	1.64	0.00	0.00
$df = 5,\ R^* = 0.3$	0.80	0.49	0.00	0.00	0.93	0.77	0.00	0.00	0.50	0.45	0.00	0.00
$df = 5,\ R^* = 0.5$	0.99	0.79	0.00	0.00	0.83	0.69	0.00	0.00	0.73	1.06	0.00	0.00
$df = 5,\ R^* = 0.7$	0.76	0.47	0.00	0.00	0.80	0.72	0.00	0.00	0.38	0.40	0.00	0.00
$df = \infty,\ R^* = 0.1$	2.05	1.82	0.00	0.00	4.80	6.08	0.00	0.00	4.89	8.52	0.00	0.00
$df = \infty,\ R^* = 0.3$	1.53	0.80	0.00	0.00	3.90	4.95	0.00	0.00	5.17	17.76	0.00	0.00
$df = \infty,\ R^* = 0.5$	2.48	3.39	0.00	0.00	5.39	5.48	0.00	0.00	6.77	14.34	0.00	0.00
$df = \infty,\ R^* = 0.7$	2.00	1.95	0.00	0.00	4.98	7.45	0.00	0.00	5.33	10.20	0.00	0.00
DI-PSO												

Obj = 3	Global nbhd				Ring-3 nbhd				Ring-1 nbhd		
Algorithm	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$	Mean	SD	$\widehat{p}_2$
PSO	144.46	295.40	0.00	0.00	35.66	54.01	0.04	0.00	25.95	68.25	0.00
BBPSO-MC	131.31	90.51	0.00	0.00	110.19	63.02	0.00	0.00	84.45	71.65	0.00
BBPSOxp-MC	37.56	14.15	0.00	0.00	25.53	8.62	0.00	0.00	18.77	6.24	0.00
AT-BBPSO-MC											
$df = 1,\ R^* = 0.1$	111.48	250.62	0.00	0.00	53.72	43.43	0.00	0.00	110.78	165.53	0.00
$df = 1,\ R^* = 0.3$	149.32	242.67	0.00	0.00	90.41	154.44	0.00	0.00	60.90	60.45	0.00
$df = 1,\ R^* = 0.5$	106.08	127.98	0.00	0.00	78.98	111.62	0.00	0.00	49.87	53.05	0.00
$df = 1,\ R^* = 0.7$	120.35	187.50	0.00	0.00	79.60	107.68	0.00	0.00	83.29	126.26	0.00
$df = 3,\ R^* = 0.1$	114.66	118.03	0.00	0.00	88.22	58.20	0.00	0.00	69.83	60.08	0.00
$df = 3,\ R^* = 0.3$	144.03	225.00	0.00	0.00	78.97	65.90	0.00	0.00	73.54	100.15	0.00
$df = 3,\ R^* = 0.5$	121.44	127.02	0.00	0.00	79.52	71.45	0.00	0.00	57.98	48.82	0.00
$df = 3,\ R^* = 0.7$	108.30	81.33	0.00	0.00	71.77	64.65	0.00	0.00	54.11	46.59	0.00
$df = 5,\ R^* = 0.1$	125.89	116.43	0.00	0.00	78.12	32.51	0.00	0.00	71.77	44.25	0.00
$df = 5,\ R^* = 0.3$	108.21	69.88	0.00	0.00	94.78	44.39	0.00	0.00	67.79	65.60	0.00
$df = 5,\ R^* = 0.5$	111.18	90.71	0.00	0.00	78.14	35.93	0.00	0.00	68.41	67.38	0.00
$df = 5,\ R^* = 0.7$	107.09	63.45	0.00	0.00	93.72	69.07	0.00	0.00	71.64	92.09	0.00
$df = \infty,\ R^* = 0.1$	126.16	71.55	0.00	0.00	99.98	25.30	0.00	0.00	68.42	43.53	0.00
$df = \infty,\ R^* = 0.3$	127.38	66.15	0.00	0.00	103.96	31.69	0.00	0.00	68.21	54.76	0.00
$df = \infty,\ R^* = 0.5$	109.58	30.42	0.00	0.00	102.75	39.42	0.00	0.00	76.03	59.73	0.00
$df = \infty,\ R^* = 0.7$	112.31	31.53	0.00	0.00	111.33	47.31	0.00	0.00	72.48	29.60	0.00
AT-BBPSOxp-MC											
$df = 1,\ R^* = 0.1$	54.91	71.36	0.00	0.00	46.43	53.25	0.00	0.00	27.74	33.78	0.00
$df = 1,\ R^* = 0.3$	42.75	27.54	0.00	0.00	34.70	44.52	0.00	0.00	26.56	35.25	0.00
$df = 1,\ R^* = 0.5$	47.05	33.70	0.00	0.00	52.38	84.94	0.00	0.00	35.21	47.68	0.00
$df = 1,\ R^* = 0.7$	44.34	44.22	0.00	0.00	55.48	108.75	0.00	0.00	39.44	46.50	0.00
$df = 3,\ R^* = 0.1$	34.88	12.52	0.00	0.00	35.55	22.20	0.00	0.00	19.04	13.27	0.00
$df = 3,\ R^* = 0.3$	40.08	14.57	0.00	0.00	33.62	25.98	0.00	0.00	23.77	19.44	0.00
$df = 3,\ R^* = 0.5$	44.76	48.60	0.00	0.00	27.92	12.79	0.00	0.00	27.27	35.43	0.00
$df = 3,\ R^* = 0.7$	46.32	77.32	0.00	0.00	30.46	14.12	0.00	0.00	18.80	8.33	0.00
$df = 5,\ R^* = 0.1$	40.72	15.41	0.00	0.00	35.09	28.26	0.00	0.00	20.00	19.18	0.00
$df = 5,\ R^* = 0.3$	40.81	22.60	0.00	0.00	32.33	19.05	0.00	0.00	20.36	12.27	0.00
$df = 5,\ R^* = 0.5$	43.59	51.50	0.00	0.00	43.91	76.14	0.00	0.00	29.54	51.03	0.00
$df = 5,\ R^* = 0.7$	38.25	13.76	0.00	0.00	29.31	12.40	0.00	0.00	21.43	26.09	0.00
$df = \infty,\ R^* = 0.1$	40.62	19.73	0.00	0.00	31.86	18.35	0.00	0.00	25.61	42.18	0.00
$df = \infty,\ R^* = 0.3$	37.54	31.58	0.00	0.00	27.73	9.34	0.00	0.00	17.79	6.45	0.00
$df = \infty,\ R^* = 0.5$	39.05	14.07	0.00	0.00	27.79	8.39	0.00	0.00	20.38	6.64	0.00
$df = \infty,\ R^* = 0.7$	35.58	12.57	0.00	0.00	28.22	8.33	0.00	0.00	18.96	7.86	0.00
DI-PSO											

Obj = 4	Global nbhd				Ring-3 nbhd				Ring-1 nbhd			
Algorithm	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$
PSO	4.32	2.75	0.02	0.02	0.65	0.89	0.56	0.56	0.13	0.47	0.90	0.86
BBPSO-MC	2.75	0.93	0.00	0.00	2.67	0.81	0.00	0.00	1.47	1.00	0.00	0.00
BBPSOxp-MC	0.13	0.05	0.00	0.00	0.04	0.02	0.04	0.00	0.01	0.00	0.86	0.00
AT-BBPSO-MC												
$df = 1,\ R^* = 0.1$	1.02	0.80	0.00	0.00	0.48	0.38	0.00	0.00	0.21	0.23	0.00	0.00
$df = 1,\ R^* = 0.3$	1.10	0.88	0.00	0.00	0.58	0.39	0.00	0.00	0.23	0.28	0.00	0.00
$df = 1,\ R^* = 0.5$	0.90	0.66	0.00	0.00	0.51	0.33	0.00	0.00	0.37	0.42	0.00	0.00
$df = 1,\ R^* = 0.7$	0.98	0.68	0.00	0.00	0.50	0.31	0.00	0.00	0.29	0.43	0.00	0.00
$df = 3,\ R^* = 0.1$	1.13	0.56	0.00	0.00	0.91	0.42	0.00	0.00	0.47	0.48	0.00	0.00
$df = 3,\ R^* = 0.3$	1.25	0.54	0.00	0.00	0.89	0.52	0.00	0.00	0.46	0.48	0.00	0.00
$df = 3,\ R^* = 0.5$	1.23	0.48	0.00	0.00	0.97	0.48	0.00	0.00	0.40	0.47	0.00	0.00
$df = 3,\ R^* = 0.7$	1.25	0.63	0.00	0.00	1.05	0.64	0.00	0.00	0.40	0.45	0.00	0.00
$df = 5,\ R^* = 0.1$	1.54	0.64	0.00	0.00	1.25	0.53	0.00	0.00	0.81	0.62	0.00	0.00
$df = 5,\ R^* = 0.3$	1.64	0.48	0.00	0.00	1.38	0.63	0.00	0.00	0.53	0.49	0.00	0.00
$df = 5,\ R^* = 0.5$	1.72	0.60	0.00	0.00	1.59	0.77	0.00	0.00	0.55	0.47	0.00	0.00
$df = 5,\ R^* = 0.7$	1.77	0.66	0.00	0.00	1.30	0.67	0.00	0.00	0.53	0.43	0.00	0.00
$df = \infty,\ R^* = 0.1$	2.63	0.78	0.00	0.00	2.42	0.86	0.00	0.00	1.38	0.86	0.00	0.00
$df = \infty,\ R^* = 0.3$	2.52	0.63	0.00	0.00	2.51	0.58	0.00	0.00	1.22	0.76	0.00	0.00
$df = \infty,\ R^* = 0.5$	2.50	0.84	0.00	0.00	2.40	0.77	0.00	0.00	1.49	0.90	0.00	0.00
$df = \infty,\ R^* = 0.7$	2.63	0.66	0.00	0.00	2.62	0.72	0.00	0.00	1.34	0.95	0.00	0.00
AT-BBPSOxp-MC												
$df = 1,\ R^* = 0.1$	0.09	0.04	0.00	0.00	0.02	0.01	0.06	0.00	0.00	0.00	1.00	0.00
$df = 1,\ R^* = 0.3$	0.08	0.04	0.00	0.00	0.03	0.01	0.02	0.00	0.00	0.00	0.94	0.00
$df = 1,\ R^* = 0.5$	0.07	0.03	0.00	0.00	0.03	0.01	0.10	0.00	0.00	0.00	0.94	0.00
$df = 1,\ R^* = 0.7$	0.07	0.03	0.00	0.00	0.03	0.01	0.04	0.00	0.00	0.00	1.00	0.00
$df = 3,\ R^* = 0.1$	0.10	0.04	0.00	0.00	0.03	0.01	0.02	0.00	0.00	0.00	0.98	0.00
$df = 3,\ R^* = 0.3$	0.10	0.04	0.00	0.00	0.03	0.01	0.02	0.00	0.00	0.00	0.98	0.00
$df = 3,\ R^* = 0.5$	0.10	0.04	0.00	0.00	0.03	0.01	0.00	0.00	0.00	0.00	1.00	0.00
$df = 3,\ R^* = 0.7$	0.09	0.04	0.00	0.00	0.03	0.02	0.06	0.00	0.00	0.00	0.96	0.00
$df = 5,\ R^* = 0.1$	0.11	0.04	0.00	0.00	0.03	0.02	0.02	0.00	0.00	0.00	0.90	0.00
$df = 5,\ R^* = 0.3$	0.10	0.04	0.00	0.00	0.03	0.01	0.00	0.00	0.00	0.00	0.96	0.00
$df = 5,\ R^* = 0.5$	0.11	0.05	0.00	0.00	0.03	0.01	0.00	0.00	0.00	0.00	0.98	0.00
$df = 5,\ R^* = 0.7$	0.10	0.04	0.00	0.00	0.03	0.02	0.02	0.00	0.00	0.00	0.94	0.00
$df = \infty,\ R^* = 0.1$	0.12	0.05	0.00	0.00	0.04	0.02	0.00	0.00	0.00	0.00	0.98	0.02
$df = \infty,\ R^* = 0.3$	0.12	0.05	0.00	0.00	0.04	0.02	0.00	0.00	0.01	0.00	0.84	0.00
$df = \infty,\ R^* = 0.5$	0.13	0.05	0.00	0.00	0.04	0.02	0.00	0.00	0.01	0.00	0.86	0.00
$df = \infty,\ R^* = 0.7$	0.12	0.04	0.00	0.00	0.04	0.02	0.00	0.00	0.00	0.00	0.94	0.00
DI-PSO												

Obj = 5	Global nbhd				Ring-3 nbhd				Ring-1 nbhd			
Algorithm	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$
PSO	0.18	0.17	0.00	0.00	0.07	0.04	0.06	0.02	0.06	0.04	0.04	0.02
BBPSO-MC	107.55	12.45	0.00	0.00	123.72	13.02	0.00	0.00	139.39	20.19	0.00	0.00
BBPSOxp-MC	161.19	16.06	0.00	0.00	147.39	15.95	0.00	0.00	88.54	22.99	0.00	0.00
AT-BBPSO-MC												
$df = 1,\ \ R^* = 0.1$	0.24	0.20	0.00	0.00	0.11	0.06	0.00	0.00	0.10	0.06	0.00	0.00
$df = 1,\ \ R^* = 0.3$	0.22	0.15	0.00	0.00	0.14	0.08	0.00	0.00	0.10	0.08	0.02	0.00
$df = 1,\ \ R^* = 0.5$	0.23	0.14	0.00	0.00	0.12	0.09	0.00	0.00	0.10	0.05	0.00	0.00
$df = 1,\ \ R^* = 0.7$	0.24	0.16	0.00	0.00	0.13	0.09	0.00	0.00	0.09	0.05	0.00	0.00
$df = 3,\ \ R^* = 0.1$	0.75	0.61	0.00	0.00	0.29	0.18	0.00	0.00	0.50	0.34	0.00	0.00
$df = 3,\ \ R^* = 0.3$	0.78	0.59	0.00	0.00	0.26	0.15	0.00	0.00	0.61	0.37	0.00	0.00
$df = 3,\ \ R^* = 0.5$	0.59	0.41	0.00	0.00	0.27	0.17	0.00	0.00	0.51	0.31	0.00	0.00
$df = 3,\ \ R^* = 0.7$	0.64	0.45	0.00	0.00	0.28	0.15	0.00	0.00	0.57	0.40	0.00	0.00
$df = 5,\ \ R^* = 0.1$	7.13	3.33	0.00	0.00	16.46	5.52	0.00	0.00	33.67	9.42	0.00	0.00
$df = 5,\ \ R^* = 0.3$	8.03	3.84	0.00	0.00	16.27	4.90	0.00	0.00	31.74	9.31	0.00	0.00
$df = 5,\ \ R^* = 0.5$	7.18	4.05	0.00	0.00	16.78	5.88	0.00	0.00	36.30	12.42	0.00	0.00
$df = 5,\ \ R^* = 0.7$	7.62	3.58	0.00	0.00	15.76	5.55	0.00	0.00	32.13	9.37	0.00	0.00
$df = \infty,\ R^* = 0.1$	108.04	13.07	0.00	0.00	122.56	10.67	0.00	0.00	138.20	17.14	0.00	0.00
$df = \infty,\ R^* = 0.3$	109.07	12.30	0.00	0.00	123.46	10.82	0.00	0.00	137.72	19.34	0.00	0.00
$df = \infty,\ R^* = 0.5$	108.41	13.09	0.00	0.00	124.54	11.65	0.00	0.00	137.44	19.45	0.00	0.00
$df = \infty,\ R^* = 0.7$	108.26	12.55	0.00	0.00	125.60	10.86	0.00	0.00	138.92	19.62	0.00	0.00
AT-BBPSOxp-MC												
$df = 1,\ \ R^* = 0.1$	0.09	0.04	0.00	0.00	0.08	0.04	0.00	0.00	0.05	0.04	0.02	0.00
$df = 1,\ \ R^* = 0.3$	0.10	0.05	0.00	0.00	0.08	0.05	0.06	0.00	0.06	0.03	0.02	0.00
$df = 1,\ \ R^* = 0.5$	0.10	0.06	0.00	0.00	0.08	0.05	0.00	0.00	0.07	0.03	0.02	0.00
$df = 1,\ \ R^* = 0.7$	0.10	0.06	0.00	0.00	0.08	0.05	0.00	0.00	0.06	0.04	0.06	0.00
$df = 3,\ \ R^* = 0.1$	3.37	2.43	0.00	0.00	4.40	2.86	0.00	0.00	0.90	0.87	0.00	0.00
$df = 3,\ \ R^* = 0.3$	3.18	2.42	0.00	0.00	3.26	1.80	0.00	0.00	1.10	1.16	0.00	0.00
$df = 3,\ \ R^* = 0.5$	3.27	2.30	0.00	0.00	3.59	3.42	0.00	0.00	0.97	1.20	0.00	0.00
$df = 3,\ \ R^* = 0.7$	3.71	2.43	0.00	0.00	4.97	4.03	0.00	0.00	0.83	0.70	0.00	0.00
$df = 5,\ \ R^* = 0.1$	64.33	10.31	0.00	0.00	62.16	10.89	0.00	0.00	25.49	10.74	0.00	0.00
$df = 5,\ \ R^* = 0.3$	63.62	10.26	0.00	0.00	61.79	12.09	0.00	0.00	26.34	10.67	0.00	0.00
$df = 5,\ \ R^* = 0.5$	62.01	10.94	0.00	0.00	62.58	9.87	0.00	0.00	26.15	13.04	0.00	0.00
$df = 5,\ \ R^* = 0.7$	64.75	10.55	0.00	0.00	62.63	11.46	0.00	0.00	25.45	10.98	0.00	0.00
$df = \infty,\ R^* = 0.1$	157.62	14.20	0.00	0.00	148.31	13.62	0.00	0.00	87.94	24.29	0.00	0.00
$df = \infty,\ R^* = 0.3$	160.30	15.23	0.00	0.00	146.65	16.52	0.00	0.00	87.96	23.07	0.00	0.00
$df = \infty,\ R^* = 0.5$	158.78	16.14	0.00	0.00	150.94	14.95	0.00	0.00	91.44	21.82	0.00	0.00
$df = \infty,\ R^* = 0.7$	162.49	16.45	0.00	0.00	145.00	16.78	0.00	0.00	87.56	20.07	0.00	0.00
DI-PSO												

Obj = 6	Global nbhd				Ring-3 nbhd				Ring-1 nbhd			
Algorithm	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$	Mean	SD	$\widehat{p}_2$	$\widehat{p}_4$
PSO	18.95	3.04	0.02	0.02	9.06	9.92	0.54	0.50	13.43	9.17	0.24	0.08
BBPSO-MC	20.25	0.10	0.00	0.00	20.28	0.12	0.00	0.00	17.53	6.84	0.00	0.00
BBPSOxp-MC	19.87	0.07	0.00	0.00	19.75	0.10	0.00	0.00	19.44	0.30	0.00	0.00
AT-BBPSO-MC												
$df = 1,\ R^* = 0.1$	5.91	9.04	0.00	0.00	0.69	2.72	0.00	0.00	0.13	0.06	0.00	0.00
$df = 1,\ R^* = 0.3$	5.56	8.81	0.00	0.00	1.10	3.96	0.00	0.00	0.92	3.99	0.00	0.00
$df = 1,\ R^* = 0.5$	5.94	8.99	0.00	0.00	0.27	0.08	0.00	0.00	0.22	0.42	0.00	0.00
$df = 1,\ R^* = 0.7$	4.05	7.69	0.00	0.00	0.64	2.85	0.00	0.00	0.59	2.89	0.00	0.00
$df = 3,\ R^* = 0.1$	18.85	4.73	0.00	0.00	8.42	9.54	0.00	0.00	7.18	9.53	0.00	0.00
$df = 3,\ R^* = 0.3$	19.44	3.81	0.00	0.00	8.01	9.68	0.00	0.00	4.67	8.33	0.00	0.00
$df = 3,\ R^* = 0.5$	19.39	3.85	0.00	0.00	10.02	9.83	0.00	0.00	5.14	8.57	0.00	0.00
$df = 3,\ R^* = 0.7$	17.86	6.44	0.00	0.00	10.36	9.94	0.00	0.00	5.23	8.48	0.00	0.00
$df = 5,\ R^* = 0.1$	20.21	0.12	0.00	0.00	17.11	7.23	0.00	0.00	10.20	9.72	0.00	0.00
$df = 5,\ R^* = 0.3$	20.19	0.12	0.00	0.00	17.47	6.81	0.00	0.00	10.47	9.85	0.00	0.00
$df = 5,\ R^* = 0.5$	20.20	0.13	0.00	0.00	17.43	6.90	0.00	0.00	8.22	9.62	0.00	0.00
$df = 5,\ R^* = 0.7$	20.22	0.13	0.00	0.00	17.81	6.34	0.00	0.00	12.09	9.80	0.00	0.00
$df = \infty,\ R^* = 0.1$	20.23	0.13	0.00	0.00	20.25	0.11	0.00	0.00	16.99	7.15	0.00	0.00
$df = \infty,\ R^* = 0.3$	20.26	0.12	0.00	0.00	20.25	0.16	0.00	0.00	16.78	7.44	0.00	0.00
$df = \infty,\ R^* = 0.5$	20.22	0.09	0.00	0.00	19.87	2.79	0.00	0.00	17.97	6.14	0.00	0.00
$df = \infty,\ R^* = 0.7$	20.24	0.10	0.00	0.00	20.23	0.13	0.00	0.00	18.02	6.09	0.00	0.00
AT-BBPSOxp-MC												
$df = 1,\ R^* = 0.1$	2.19	5.99	0.00	0.00	0.49	2.73	0.00	0.00	1.00	4.02	0.04	0.00
$df = 1,\ R^* = 0.3$	1.41	4.69	0.00	0.00	1.28	4.75	0.00	0.00	0.07	0.16	0.04	0.00
$df = 1,\ R^* = 0.5$	1.11	3.97	0.00	0.00	0.14	0.39	0.00	0.00	0.06	0.15	0.00	0.00
$df = 1,\ R^* = 0.7$	0.63	2.81	0.00	0.00	0.53	2.66	0.00	0.00	0.54	2.87	0.04	0.00
$df = 3,\ R^* = 0.1$	18.23	5.28	0.00	0.00	14.49	8.28	0.00	0.00	11.62	9.28	0.00	0.00
$df = 3,\ R^* = 0.3$	18.68	4.03	0.00	0.00	14.48	8.29	0.00	0.00	9.35	9.18	0.00	0.00
$df = 3,\ R^* = 0.5$	18.17	5.21	0.00	0.00	15.52	7.62	0.00	0.00	8.76	9.22	0.02	0.00
$df = 3,\ R^* = 0.7$	15.52	7.99	0.00	0.00	15.31	7.87	0.00	0.00	11.95	8.96	0.00	0.00
$df = 5,\ R^* = 0.1$	19.39	2.12	0.00	0.00	19.35	2.68	0.00	0.00	15.93	7.26	0.00	0.00
$df = 5,\ R^* = 0.3$	19.82	0.10	0.00	0.00	19.58	0.28	0.00	0.00	16.12	6.99	0.00	0.00
$df = 5,\ R^* = 0.5$	19.43	2.68	0.00	0.00	19.35	2.42	0.00	0.00	17.39	5.38	0.00	0.00
$df = 5,\ R^* = 0.7$	19.81	0.12	0.00	0.00	19.29	2.74	0.00	0.00	15.17	7.86	0.00	0.00
$df = \infty,\ R^* = 0.1$	19.85	0.08	0.00	0.00	19.73	0.12	0.00	0.00	18.69	3.37	0.00	0.00
$df = \infty,\ R^* = 0.3$	19.87	0.07	0.00	0.00	19.75	0.11	0.00	0.00	18.88	2.97	0.00	0.00
$df = \infty,\ R^* = 0.5$	19.85	0.09	0.00	0.00	19.76	0.11	0.00	0.00	18.63	3.40	0.00	0.00
$df = \infty,\ R^* = 0.7$	19.86	0.08	0.00	0.00	19.77	0.10	0.00	0.00	19.47	0.30	0.00	0.00
DI-PSO												

References

Andrieu, C. and Thoms, J. (2008). "A tutorial on adaptive MCMC." *Statistics and Computing*, 18, 4, 343–373.

Blum, C. and Li, X. (2008). "Swarm Intelligence in Optimization." In *Swarm Intelligence: Introduction and Applications*, eds. C. Blum and D. Merkle. Springer.

Clerc, M. (2010). *Particle swarm optimization*. John Wiley & Sons.

Clerc, M. and Kennedy, J. (2002). "The particle swarm-explosion, stability, and convergence in a multidimensional complex space." *Evolutionary Computation, IEEE Transactions on*, 6, 1, 58–73.

Gelman, A., Roberts, G., and Gilks, W. (1996). "Efficient Metropolis jumping rules." *Bayesian statistics*, 5, 599-608, 42.

Hsieh, H.-I. and Lee, T.-S. (2010). "A modified algorithm of bare bones particle swarm optimization." *International Journal of Computer Science Issues*, 7, 11.

Magnus, J. R. (1988). *Linear structures*. No. 42 in Griffin's Statistical Monographs and Courses. New York: Oxford University Press.

Magnus, J. R. and Neudecker, H. (1980). "The elimination matrix: some lemmas and applications." *SIAM Journal on Algebraic Discrete Methods*, 1, 4, 422–449.

— (2005). *Matrix differential calculus with applications in statistics and econometrics*. 3rd ed. New York: John Wiley & Sons.

Zhang, H., Kennedy, D. D., Rangaiah, G. P., and Bonilla-Petriciolet, A. (2011). "Novel bare-bones particle swarm optimization and its performance for modeling vapor–liquid equilibrium data." *Fluid Phase Equilibria*, 301, 1, 33–45.