8 Numerical Methods

8.1 Optimization Methods

Some of the estimators discussed in the previous chapters cannot be derived analytically, but are implicitly defined as the solution to an optimization problem: $\hat{\theta} = \arg\min_{\theta \in \Theta} Q(\theta, Y^n)$. Chapter 6 dealt with the asymptotic properties of such estimators.[1] This section sketches the numerical algorithms that can be used to obtain $\hat{\theta}$. The goal is to give some intuition on the classes of methods available rather than to provide recipes to be implemented one-to-one, because every statistical software contains many such optimization methods as pre-programmed commands and advanced methods are available as external software.

Gradient-Based Iterative Algorithms Numerous optimization algorithms are based on an iterative updating of a current guess θ^m to obtain a new guess θ^{m+1} until, hopefully, the optimum is reached. They are motivated by the follwing mean-value expansion of the score of the objective function (vector of first-order derivatives), $Q^{(1)}(\theta, Y^n) = \frac{\partial Q(\theta, Y^n)}{\partial \theta}$, around θ^m:

$$Q^{(1)}(\theta^{m+1}, Y^n) \approx Q^{(1)}(\theta^m, Y^n) - Q^{(2)}(\theta^m, Y^n)(\theta^{m+1} - \theta^m) \ ,$$

where $Q^{(2)}(\theta, Y^n) = \frac{\partial^2 Q(\theta, Y^n)}{\partial \theta \partial \theta'}$ is the Hessian of the objective function (matrix of second-order derivatives). If $Q(\theta, Y^m)$ is a quadratic function, this relationship is exact. For θ^{m+1} to be the optimum – provided that it is interior – $Q^{(1)}(\theta^{m+1}, Y^n) = 0$ must hold. This yields the expression

$$\theta^{m+1} = \theta^m - Q^{(2)}(\theta^m, Y^n)^{-1} Q^{(1)}(\theta^m, Y^n) \ ,$$

which can be used to update guess θ^m to guess θ^{m+1}.

The Newton algorithm uses exactly this equation as-is:

[1] For this purpose, we wrote this (sample) objective function as Q_n to distinguish it from the population or limit objective function.

Algorithm 4 (Newton Algorithm)**.**

1. *Initialize θ^0 and specify a tolerance level $\epsilon > 0$.*
2. *For $m = 1, 2, ...,$ do the following:*
   (a) Given θ^m, compute $Q^{(2)}(\theta^m, Y^n)^{-1}$ and $Q^{(1)}(\theta^m, Y^n)$ and set

$$\theta^{m+1} = \theta^m - Q^{(2)}(\theta^m, Y^n)^{-1} Q^{(1)}(\theta^m, Y^n) \, .$$

   (b) If $||(\theta^{m+1} - \theta^m|| < \epsilon$, take $\hat{\theta} = \theta^{m+1}$. Else, proceed to the next iteration.

Instead of checking convergence based on the change in the parameter value θ from iteration m to iteration $m + 1$, it is also common to assesss the change in the objective function $Q(\theta, Y^n)$ or to check whether the score $Q^{(1)}(\theta, Y^n)$ is sufficiently close to zero.

If $Q(\theta, Y^m)$ was a quadratic function, the Newton algorithm would converge to the optimum in a single iteration. Even otherwise, the algorithm tends to work very well. A drawback of the Newton algorithm is that the inverse of the Hessian needs to be computed in every step, which is often computationally expensive. As a result, many other iterative optimization algorithms replace the Hessian with some other matrix that is easier to compute. For example, one might use the score squared or the expectation thereof, as motivated by the information matrix equality. The simplest approach is to just take the identity matrix, as in the Method of Steepest Descent.

Algorithms that build on the Newton algorithm also often specify a step size $r > 0$ that multiplies the updating-term $Q^{(2)}(\theta^m, Y^n)^{-1} Q^{(1)}(\theta^m, Y^n)$. The rationale is that taking $r = 1$ as in the Newton algorithm might i) imply too large updating steps, leading us to miss the optimum (obtain an increase in the objective function) or to cycle around the optimum, or ii) it might imply too small steps and therefore slow convergence. Some algorithms start every iteration with $r = 1$ and, if the objective function (or the absolute value of the score) increases, decrease r by enough such that it increases instead. Other algorithms choose r adaptively using some criterion, thereby updating r^m to r^{m+1} in every iteration.

Importantly, for non-convex objective functions, there is no guarantee that a numerical optimization procedure finds the global optimum. It is good practice to run the algorithm for many different initial values θ^0 and take the lowest value of all of these local minima as the final solution.

Iteration on Conditional Optimizers Sometimes, there is no closed form solution for the estimator entire parameter vector θ, but we can obtain analytical expressions for conditional estimators (optimizers). For example, consider the GLS model from Section 3.4. Under Normality of errors, it leads to the likelihood

$$\begin{aligned} p(Y|\beta,\Sigma) &= \prod_{i=1}^{n}(2\pi\sigma_i^2)^{-\frac{1}{2}} exp\left\{-\frac{1}{2\sigma_i^2}(y_i - x_i'\beta)^2\right\} \\ &= (2\pi)^{-\frac{n}{2}}|\Sigma|^{-\frac{1}{2}} exp\left\{-\frac{1}{2}(Y - X\beta)'\Sigma^{-1}(Y - X\beta)\right\} , \end{aligned}$$

where $\Sigma = diag(\sigma_1^2, ..., \sigma_n^2)$.The joint estimator $\hat{\theta} = (\hat{\beta}, \hat{\Sigma}) = \arg\max_{\theta=(\beta,\Sigma)} p(Y|\beta,\Sigma)$ is not available analytically, but we can solve for the conditional estimators

$$\hat{\beta}|\Sigma = (X'\Sigma^{-1}X)^{-1}X'\Sigma^{-1}Y , \quad \text{and} \quad \hat{\Sigma}|\beta = diag\left\{(Y - X\beta)(Y - X\beta)'\right\} .^2$$

Note that the same conditional estimators are obtained under LS estimation:

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i'\beta)^2/\sigma_i^2 = \arg\min_{\beta} \frac{1}{n}(Y - X\beta)\Sigma^{-1}(Y - X\beta) ,$$

and in turn we set $\hat{\sigma}_i^2 = (y_i - x_i'\beta)^2$ for $i = 1 : n$, which corresponds to the above expression for $\hat{\Sigma}|\beta$.

One approach to obtain an estimator for $\hat{\theta} = (\hat{\beta}, \hat{\Sigma})$ is two-step GLS estimation, as discussed in Section 3.4:

1. Set $\Sigma = 1$ and let $\tilde{\beta}|(\Sigma = 1)$ be a preliminary estimate for β. Note that this amounts to OLS estimation.
2. Conditioning on this $\tilde{\beta}$ from the first step, compute $\tilde{\Sigma}|\tilde{\beta}$ and and use it to obtain the "feasible" GLS estimator $\hat{\beta}|\ \tilde{\Sigma}$. The final estimate of Σ is then given by $\hat{\Sigma}|\hat{\beta}$.

This estimator does not correspond to $\arg\max_{\beta,\Sigma} p(Y|\beta,\Sigma)$, but it still has good properties.[3]

We can obtain the estimator $\hat{\theta} = (\hat{\beta}, \hat{\Sigma}) = \arg\max_{\theta=(\beta,\Sigma)} p(Y|\beta,\Sigma)$ by iterating on the two conditional estimators above until convergence (see Meng and Rubin (1993)):

[2] i.e. $\hat{\sigma}_i^2 = (y_i - x_i'\beta)^2$.

[3] Recall from Chapter 3 that OLS is consistent even under heteroskedasticity, which in turn implies that $\tilde{\Sigma}$ is consistent (plug-in property). In turn, $\hat{\beta}|\ \tilde{\Sigma}$ is consistent thanks to consistency of $\tilde{\Sigma}$ (see Section 6.2.1), which in turn implies that $\hat{\Sigma}$ is consistent. The asymptotic variance of $\hat{\beta}$ needs to be adjusted for the fact that $\hat{\Sigma}$ is estimated (see Section 6.2.1).

Algorithm 5 (Meng and Rubin (1993) Algorithm for Two Parameter Groups, β and Σ)**.**

1. *Initialize* Σ^0 *– e.g.* $\Sigma^0 = I$ *– and specify tolerance level* $\epsilon > 0$.
2. *For* $m = 0, 1, 2, ...,$ *given* Σ^m,
   (a) *compute* $\beta^{m+1} = \hat{\beta}|\Sigma^m$,
   (b) *compute* $\Sigma^{m+1} = \hat{\Sigma}|\beta^{m+1}$,
   (c) *if* $||(\theta^{m+1} - \theta^m|| < \epsilon$, *take* $\hat{\theta} = \theta^{m+1}$. *Else, proceed to the next iteration.*

The Appendix states the algorithm for a generic parameter vector θ and a general partition of it.

8.2 Sampling Methods

Often in econometrics, we need to sample random numbers from a certain distribution. For example, to evaluate the properties of an estimator or test statistic in some model, we may want to sample data from the supposed distribution that characterizes the model (see Chapter 2 or Section 7.1). In addition, in Bayesian analysis, we oftentimes cannot obtain the posterior $p(\theta|Y)$ analytically, but rely on numerical sampling techniques. While sampling from standard distributions can be done easily and efficiently using commands from statistical software, we sometimes need to sample from some distribution specific to our application, e.g. a non-standard posterior. This section presents methods that we can use in these cases.

The random sampling commands in statistical software are based on a sample of uniform random numbers. It turns out that solutions to certain non-linear deterministic difference equations are indistinguishable from $U(0, 1)$ random numbers. Thereby, the so-called "seed" determines the initial condition of these difference equations and hence influences the (quasi-) random number we get. It is very important to set the seed at the beginning of any code that involves random sampling, as it ensures that the code always gives the same random numbers, which renders the analysis replicable.

Inverse CDF Method Suppose we want to draw a random number θ with pdf $f(\theta)$ and cdf $F(\theta)$. We can turn a Uniformly distributed random number, $\theta^* \sim U(0,1)$, into such a random number with pdf $f(\theta)$ by setting $F(\theta) = \theta^*$ and solving for θ.

Algorithm 6 (Inverse CDF Method)**.**
For $m = 1 : M$
1. *Draw* $\theta^* \sim U(0,1)$.
2. *Set* $\theta^m = F^{-1}(\theta^*)$.

For example, for an exponential distribution, we have $f(\theta) = \lambda e^{-\lambda\theta}$ and $F(\theta) = -e^{-\lambda\theta} + 1$, whereby $\lambda > 0$ and $\theta \geqslant 0$. We can get an exponentially distributed random number as $\theta = F^{-1}(\theta^*) = -\frac{1}{\lambda}\ln(1-\theta^*)$ for $\theta^* \sim U(0,1)$.

Accept-Reject Method Suppose we want to draw from $f(\theta)$, but we only know how to draw from $g(\theta)$. The accept-reject method generates draws from $f(\theta)$ by sampling from $g(\theta)$ and only accepting the draw with some probability. This probability is higher if the draw has a lot of probability mass under the pdf f relative to the pdf g.

Algorithm 7 (Accept-Reject Method)**.**

1. *Fix* c *s.t.* $\dfrac{f(\theta)}{g(\theta)} \leqslant c < \infty \quad \forall\, \theta$.
2. *Repeat the following steps until you sampled* M *random numbers:*
   (a) *draw proposal* $\theta^* \sim g(\theta)$,
   (b) *accept* $\theta = \theta^*$ *with probability* $\dfrac{f(\theta^*)}{cg(\theta^*)}$, *else discard* θ^*.

The resulting set of draws, $\{\theta^m\}_{m=1}^M$ will be mutually independent. Relatedly, the algorithm is easily parallelizable as it is not sequential, i.e. in each iteration we draw a separate value, unaffected by the other draws (iterations). Also, note that we only need to know f and g up to proportionality, i.e. any constants get absorbed into c. For example, under Bayesian estimation, f could be the posterior, which we often do not know analytically, but can evaluate it up to a scaling constant as the product of likelihood and prior: $f(\theta) = p(\theta|Y) \propto p(Y|\theta)p(\theta)$.

Importance Sampling Again, suppose we want to drow from $f(\theta)$, but we only know how to draw from $g(\theta)$. The importance sampling method is similar to the accept-reject method, except that it accepts all draws from $g(\theta)$ and weighs them according to the likelihood of obtaining the particular draw under f relative to g. Hence, rather than discarding a draw from g that is unlikely under f, it assigns a low weight to it.

Algorithm 8 (Importance Sampling)**.**

1. *For $m = 1 : M$, draw $\theta^m \sim g(\theta)$ and compute the importance weight $w^m = \frac{f(\theta^m)}{cg(\theta^m)}$ for some (any) constant c.*
2. *Compute the normalized weights $W^m = \frac{w^m}{\frac{1}{M}\sum_{m=1}^{M} w^m}$, $m = 1 : M$.*

The output of this algorithm is a particle approximation of f, $\{\theta^m, W^m\}_{m=1}^M$, i.e. a set of so-called "particles": values of θ with corresponding weights that sum to M. They approximate the distribution f in the sense that

$$\frac{1}{M}\sum_{m=1}^{M} W^m h(\theta^m) \approx \mathbb{E}_f[h(\theta)] = \int h(\theta) f(\theta) d\theta \ .$$

As before, the algorithm is easily parallelizable. Also, as before, only the kernels of f and g are needed. Any constants get absorbed into c, and c gets cancelled in the calculation for $\{W^m\}_{m=1}^M$.

In theory, the algorithm works for any g that has the same domain as f. However, the accuracy of the approximation is higher the higher the overlap between the distributions f and g and hence the lower the variation in weights $\{W^m\}_{m=1}^M$. Therefore, for efficiency, one wants to choose a g as close as possible to f. In particular, g should have fatter tails than f so that there is an upper bound for the weights. Otherwise the variance of weights can get very high and our approximation very imprecise.

A good choice for g is obtained by centering a fat-tailed t-distribution at the mode of f and setting the variance to the scaled-up inverse Hessian of f (or $\log f$), evaluated at the mode of f. This roughly aligns the contours of the two distributions and ensures that g has fatter tails than f. As long as f can be evaluated (up to proportionality), the mode of f can be computed by numerical optimization and the Hessian can be computed by numerical differentiation.

Gibbs Sampling Suppose we want to draw from $p(\theta)$ and we can partition $\theta = (\theta_1', \theta_2')'$ s.t. we know how to sample from the conditional distributions $p(\theta_1|\theta_2)$ and $p(\theta_2|\theta_1)$. We can use the Gibbs sampling algorithm to obtain a set of draws from the joint distribution $p(\theta)$ by iteratively drawing from the two conditionals.

As an example, consider the posterior $p(\beta, \Sigma|Y)$ under the GLS model from Section 3.4, also discussed in Section 8.1 above. The likelihood is

$$\begin{aligned} p(Y|\beta,\Sigma) &= \prod_{i=1}^{n}(2\pi\sigma_i^2)^{-\frac{1}{2}} exp\left\{-\frac{1}{2\sigma_i^2}(y_i - x_i'\beta)^2\right\} \\ &= (2\pi)^{-\frac{n}{2}}|\Sigma|^{-\frac{1}{2}} exp\left\{-\frac{1}{2}(Y - X\beta)'\Sigma^{-1}(Y - X\beta)\right\} , \end{aligned}$$

where $\Sigma = diag(\sigma_1^2, ..., \sigma_n^2)$. Under the priors $\beta \sim N(\underline{\beta}, \underline{V})$ and $\sigma_i^2 \sim IG(\underline{\nu}, \underline{s}^2)$ for $i = 1 : n$, we get the conditional posteriors

$$\beta|Y,\Sigma \sim N(\bar{\beta}, \bar{V}) , \quad \text{with} \quad \bar{V} = \left[X'\Sigma^{-1}X + \underline{V}^{-1}\right]^{-1} , \quad \bar{\beta} = \bar{V}\left[X'\Sigma^{-1}Y + \underline{V}^{-1}\underline{\beta}\right] ,$$

and

$$\sigma_i^2|Y,\beta \sim IG(\bar{\nu}, \bar{s}^2) , \quad \text{with} \quad \bar{\nu} = \underline{\nu} + 1 , \quad \bar{s}^2 = \underline{s}^2 + (y_i - x_i'\beta)^2 .$$

We cannot solve for the joint posterior $p(\beta, \Sigma|Y)$ analytically, but we can obtain it numerically by iterating on these two (sets of) conditional posteriors:[4]

Algorithm 9 (Gibbs Sampling for Two Parameter Groups, β and Σ)**.**

1. *Initialize* Σ^0 *(e.g. take* $\Sigma^0 = I$*).*
2. *For* $m = 0, 1, 2, ..., M^*$*, given* Σ^m*,*
   (a) draw β^{m+1} *from* $p(\beta|Y, \Sigma^m)$*,*
   (b) draw Σ^{m+1} *from* $p(\Sigma|\beta^{m+1})$*,*

As $M^* \to \infty$, the final draw $\theta^{M^*} = (\beta^{M^*}, \Sigma^{M^*})$ comes from the desired distribution, the joint posterior $p(\beta, \Sigma|Y)$. Based on this result, we can let the above algorithm run for some large M^* and take the last M draws, $\{\theta^{M^*-M+m}\}_{m=1}^{M}$ for $\theta = (\beta, \Sigma)$, to approximate $p(\beta, \Sigma|Y)$. The first $M^* - M$ draws are discarded because they depend on the initialization (we "burn-in" the algorithm before actually sampling). We can choose M^* – i.e. assess convergence – by verifying that the moments of the last M draws have stabilized.

[4]Note that the conditional posteriors $\{p(\sigma_i^2|Y, \beta)\}_{i=1:n}$ are independent of one another. They can be put together as a single conditional posterior $p(\Sigma|Y, \beta)$.

Note that this algorithm is not parallelizable, as the draw of every β depends on the draw of Σ from the previous iteration.[5] Relatedly, the resulting draws are not independent but follow a so-called Markov chain (every draw depends on the previous draw) whose limit distribution is the distribution of interest, $p(\beta, \Sigma|Y)$.[6] A more general version of the Gibbs sampling algorithm is presented in the Appendix.

Metropolis Hastings Algorithm The Metropolis Hastings (MH) algorithm belongs to the wider class of Markov Chain Monte Carlo (MCMC) methods, which create a sequence of serially correlated draws whose distribution converges to the distribution of interest $f(\theta)$, just as done by Gibbs sampling. However, unlike Gibbs sampling, the MH alogrithm does not require knowing conditional distributions, nor does it require having a good proposal density for f, as demanded under importance sampling.[7] [8] The only requirement is that one needs to be able to evaluate the distribution $f(\theta)$ up to proportionality. In the context of posterior sampling, this means that one needs to be able to evaluate the likelihood and prior – as $p(\theta|Y) \propto p(Y|\theta)p(\theta)$ –, which is satisfied in the vast majority of applications. The MH algorithm exploits the fact that knowing a distribution up to proportionality allows us to tell which among two draws θ and θ^* is more likely under that distribution.[9]

In the following, a particular version of the RH algrithm is discussed: the Random Walk MH (RWMH) algorithm. A more general, generic MH algorithm is discussed in the Appendix. The exposition is based on the more detailed discussion in Herbst and Schorfheide (2015).[10]

[5] Also, the draw of Σ depends on the draw of β from the same iteration.

[6] This is why, among the last draws returned by the algorithm, researchers sometimes consider only every kth draw in order to reduce the dependence of the draws on one another in the reslting sample. However, as M becomes large, whether this is done or not becomes irrelevant, as every region of the domain is visited by the algorithm often enough and in proportionality with the probability mass in that region.

[7] This makes it useful in models where we have little knowledge on where the posterior roughly lies. This increased power comes at the cost of having serially correlated draws, unlike in importance sampling.

[8] The importance sampler is rarely used to obtain the posterior. Nevertheless, it is discussed in these notes because it provides an intuition useful to understand other sampling methods and it is a building block of the Sequential Monte Carlo (SMC) algorithm. Together with the MH algorithm, the latter is one of the two methods most often encountered in practice for Bayesian estimation.

[9] Note that this is reminiscent of the accept-reject sampler.

[10] In particular, they discuss many upgrades of the basic version of the algorithm presented here.

Algorithm 10 (Random Walk MH (RWMH) Algorithm)**.**

1. *Initialize* θ^0.
2. *For* $m = 0 : M^*$, *given* θ^m, *obtain* θ^{m+1} *as follows:*
   (a) *draw a candidate* θ^* *from proposal density* $N(\theta^m, c^2\Sigma)$,
   (b) *accept the candidate with probability*

$$\alpha(\theta^*|\theta^m) = \min\left\{1, \frac{p(Y|\theta^*)p(\theta^*)}{p(Y|\theta^{i-1})p(\theta^{i-1})}\right\},$$

else continue with $\theta^{m+1} = \theta^m$. *That is, set* $\theta^{m+1} = \begin{cases} \theta^* & w.p.\ \alpha(\theta^*|\theta^m) \\ \theta^m & w.p.\ 1 - \alpha(\theta^*|\theta^m) \end{cases}$.

Just as with Gibbs sampling, we take M^* large and consider the last M draws to approximate our posterior of interest (see above for details). Note that $\alpha(\theta^*|\theta^m)$ is equal to the relative posterior probability of candidate θ^* and previous draw θ^m, $p(\theta^*|Y)/p(\theta^m|Y)$, capped to one so that it can be interpreted as a probability. Hence, the RWMH algorithm accepts a candidate θ^* with higher probability when it has higher posterior density than the previous draw. As a result, the algorithm visits regions with high posterior probability mass more often. Nevertheless, even candidates with low probability mass are occasionally accepted, with the goal of visiting all regions (which have positive probability mass) in the correct proportions.

A good choice for the proposal variance $c^2\Sigma$ is the inverse (negative) Hessian evaluated at the mode of $p(\theta|Y)$, which can be computed numerically. Alternatively, one can first let Σ be the prior variance (i.e. the covariance matrix of θ under the prior), run the algorithm to sample a small number of draws, and then set Σ to be the covariance matrix of these preliminary draws. This can be repeated multiple times, but eventually Σ needs to be fixed to guarantee convergence. Relatedly, c can be chosen adaptively to target a certain acceptance ratio, whereby a value of roughly 0.25 is shown to perform best for a wide variety of applications in the sense that it gives the most accurate approximation of statistics of interest under the desired density.[11] [12]

[11] This is assessed either by knowing the true density or by computing the variance of the computed statistics under repeated runs of the algorithm.

[12] If the sequence of draws is very "edgy", one is rejecting a lot of draws, which means that the step sizes are too large, i.e. c is too high (or q is a poor choice). If the sequence of draws is very persistent, one is accepting a lot of draws, which indicates that step sizes are too small, i.e. c is too low (the algorithm is then inefficient because it barely moves; distant regions are visited too rarely).

For efficiency reasons, it is important to store in each iteration m not only the particle value θ^m, but also the values of the likelihood and prior evaluated at θ^m, which are needed in the next iteration and might be costly to compute anew. Moreover, if the denominator is close to zero, α cannot be computed by applying the above formula as-is (software returns "not-a-number"). In these cases, we can compute it as

$$\frac{p(Y|\theta^*)p(\theta^*)}{p(Y|\theta^{i-1})p(\theta^{i-1})} = exp\left\{\log p(Y|\theta^*) + \log p(\theta^*) - \log p(Y|\theta^{i-1}) - \log p(\theta^{i-1})\right\} ,$$

which is numerically much more stable. Thereby, rather than computing a density and then taking logs, it is better to directly compute the log-density.[13]

Sequential Monte Carlo Algorithm The MH algorithm works well unless the posterior distribution of interest is multi-modal. In that case, it is unlikely that the MH algorithm, starting from one of the modes, accepts a draw that lies in the area of low probability mass between the modes in order to get to the other mode(s). As a result, one obtains a sample with disproportionately many draws from regions close to some modes and insuifficiently from regions close to other modes.[14] For such oddly-shaped posteriors, the Sequential Monte Carlo (SMC) algorithm is preferred. It has the additional advantages that it is parallelizable, which is particularly useful for models for which it is computationally costly to evaluate the likelihood. The SMC algorithm is discussed in the Appendix.

Appendix

Iteration on Conditional Estimators: General Case Suppose we are interested in $\hat{\theta}$, the joint optimizer of some objective function. Suppose further that we can partition θ into elements $\{\theta_1, ..., \theta_K\}$ s.t. we have analytical expressions for the conditional optimizers $\hat{\theta}_j|\{\theta_k\}_{k=1:K,k\neq j}$ for all $j = 1 : K$. Then we can obtain $\hat{\theta}$ using the following algorithm:

[13]For example, the pdf of a $N(0,1)$, $p(\theta) = (2\pi)^{-1/2}exp\left\{-\frac{1}{2}\theta^2\right\}$, returns values close to zero for extreme ralizations of θ. Even though $p(\theta) > 0 \ \forall \ \theta$, due to numerical imprecision statistical software might simply return a zero, in which case computing the log is not possible or gives $-\infty$. Even for very low values of $p(\theta)$, taking the log might result in $-\infty$. In contrast, $\log p(\theta) = -\frac{1}{2}\log 2\pi - \frac{1}{2}\theta^2$ can be computed for much more extreme realizations of θ without yielding $-\infty$.

[14]Except if the likelihood evaluation is so fast that one can sample enough draws to circumvent this issue.

Algorithm 11 (Meng and Rubin (1993))**.**

1. *Initialize* $\theta^0 = \{\theta_1^0, ..., \theta_K^0\}$ *and specify* $\epsilon > 0$.
2. *For* $m = 0, 1, 2, ...,$ *given* $\theta^m = \{\theta_1^m, ..., \theta_K^m\}$,
   (a) *set* $\theta_1^{m+1} = \hat{\theta}_1 | \{\theta_2^m, ..., \theta_K^m\}$,
   (b) *compute* $\theta_2^{m+1} = \hat{\theta}_2 | \{\theta_1^{m+1}, \theta_3^m, ..., \theta_K^m\}$,
   (c) *continue with remaining* $j = 3 : K$ *in the same fashion:*

$$\theta_j^{m+1} = \hat{\theta}_j | \left\{ \{\theta_k^{m+1}\}_{k=1:j-1}, \{\theta_k^m\}_{k=j+1:K} \right\} .$$

   (d) *If* $||(\theta^{m+1} - \theta^m|| < \epsilon$, *take* $\hat{\theta} = \theta^{m+1}$. *Else, proceed to the next iteration.*

Under some regularity conditions,[15] this algorithm is guaranteed to converge to the joint, unconditional optimizer of the optimization problem, $\hat{\theta}$.

Gibbs Sampling (Iteration on Draws from Conditional Distributions): General Case Suppose you want to draw from $p(\theta)$ and you can partition $\theta = [\theta_1', ..., \theta_K']'$ s.t. you can draw from $p(\theta_j | \theta_1, ..., \theta_{j-1}, \theta_{j+1}, ..., \theta_K)$, for $j = 1 : K$. Draws from $p(\theta)$ can be obtained as follows:

Algorithm 12 (Gibbs Sampling)**.**

1. *Initialize* $\theta_1^0, ..., \theta_K^0$.
2. *For* $m = 0, 1, 2, ..., M^*$, *given* θ^m, *draw* θ^{m+1} *as follows:*
   (a) *draw* θ_1^{m+1} *from* $p(\theta_1 | \theta_2^m, ..., \theta_K^m)$,
   (b) *draw* θ_2^{m+1} *from* $p(\theta_2 | \theta_1^{m+1}, \theta_3^m, ..., \theta_K^m)$, *and*
   (c) *for* $j = 3, 4, ..., K$, *draw* θ_j^{m+1} *from* $p(\theta_j | \theta_1^{m+1}, ..., \theta_{j-1}^{m+1}, \theta_{j+1}^m, ..., \theta_K^m)$.

The same comments about M^* (convergence assessment), burn-in and dependence of draws apply as in the main text.

[15] Importantly, they require the domain of every $\hat{\theta}_j | \{\theta_k\}_{k=1:K, k \neq j}$ to be the same regardless of the values for $\{\theta_k\}_{k=1:K, k \neq j}$.

Generic Metropolis Hastings algorithm The Random Walk MH (RWMH) algorithm discussed in the main text takes $N(\theta^m, c^2\Sigma)$, a Normal centered at the current draw, θ^m, as the proposal density. A more general class of MH algorithms can be obtained by considering other proposal distributions.

Algorithm 13 (Generic Metropolis Hastings (MH) Algorithm)**.**

1. *Initialize* θ^0.
2. *For* $m = 1 : M^*$*, given* θ^m*, obtain* θ^{m+1} *as follows:*
   (a) *draw a candidate* θ^* *from proposal density* $q(\theta^*|\theta^m)$*,*
   (b) *accept the candidate with probability*

$$\alpha(\theta^*|\theta^m) = \min\left\{1, \frac{p(\theta^*|Y)/q(\theta^*|\theta^m)}{p(\theta^m|Y)/q(\theta^m|\theta^*)}\right\} = \min\left\{1, \frac{p(Y|\theta^*)p(\theta^*)/q(\theta^*|\theta^m)}{p(Y|\theta^m)p(\theta^m)/q(\theta^m|\theta^*)}\right\},$$

else continue with $\theta^{m+1} = \theta^m$*. That is, set* $\theta^{m+1} = \begin{cases} \theta^* & w.p.\ \alpha(\theta^*|\theta^m) \\ \theta^m & w.p.\ 1 - \alpha(\theta^*|\theta^m) \end{cases}$.

Different specifications of the proposal density q lead to different MH algorithms. Note that q only determines how to go from one point in the parameter space to another. Therefore, its specification is much less consequential than choosing the proposal density g for the distribution of interest f (or $p(\theta|Y)$) under importance sampling.

Sequential Monte Carlo (SMC) algorithm The following exposition is based on the more detailed discussion in Herbst and Schorfheide (2015).

For notational simplicity, let $\pi(\theta) = p(\theta|Y)$. In order to draw from $\pi(\theta)$, the SMC algorithm uses a sequence of "bridge posterior distributions" $\{\pi_n(\theta)\}_{n=0}^{N_\phi}$, where the last one in the sequence equals the posterior distribution – $\pi_{N_\phi}(\theta) = \pi(\theta)$ – and where each $\pi_{n-1}(\theta)$ is used as the proposal density for $\pi_n(\theta)$, like in importance sampling. Whereas importance sampling attempts to go from some proposal distribution π_0 in a single step to the target distribution π, the SMC algorithm moves uses from π_0 to π in many steps by constructing "in-between" distributions. These bridge posteriors are constructed a sequence of "likelihood functions" $p_n(Y|\theta)$ for $n = 1 : N_\phi$ (more on them below) as

$$\pi_n(\theta) = \frac{p_n(Y|\theta)p(\theta)}{\int p_n(Y|\theta)p(\theta)d\theta}.$$

Each density $\pi_n(\theta)$ is represented by a particle approximation $\{\theta_n^i, W_n^i\}_{i=1}^N$. Thus, at stage

n the algorithm propagates the particles $\{\theta_{n-1}^i, W_{n-1}^i\}_{i=1}^N$ so that they come to represent the target density $\pi_n(\theta)$.

Algorithm 14 (Generic Sequential Monte Carlo (SMC) Algorithm)**.**

1. *Initialize the particles: obtain* $\{\theta_0^i, W_0^i\}_{i=1}^N$ *by drawing* $\theta_0^i \sim \pi_0(\theta)$ *and setting* $W_0^i = 1$ *for* $i = 1 : N$.
2. *For* $n = 1 : N_\phi$,
   (a) **Correction:** *Reweight the particles from stage* $n-1$ *by defining the incremental weights*
   $$\tilde{w}_n^i = \frac{p_n(Y|\theta_{n-1}^i)}{p_{n-1}(Y|\theta_{n-1}^i)}$$
   and the normalized weights
   $$\tilde{W}_n^i = \frac{\tilde{w}_n^i W_{n-1}^i}{\frac{1}{N}\sum_{i=1}^N \tilde{w}_n^i W_{n-1}^i}, \quad i = 1, \ldots, N.$$
   (b) **Selection (Optional):** *Resample the swarm of particles* $\{\theta_{n-1}^i, \tilde{W}_n^i\}_{i=1}^N$ *and denote resampled particles by* $\{\hat{\theta}_n^i, W_n^i\}_{i=1}^N$, *where* $W_n^i = 1 \; \forall \; i$.
   (c) **Mutation:** *For* $i = 1 : N$, *propagate particle* i, $\{\hat{\theta}_n^i, W_n^i\}$, *using* N_{MH} *steps of a MH algorithm with target distribution* $\pi_n(\theta)$. *Denote the mutated particles by* $\{\theta_n^i, W_n^i\}_{i=1}^N$.

The correction step only changes the particle weights, not the values. It is an importance sampling step; you have N draws $\{\theta_{n-1}^i\}_{i=1}^N$ from distribution π_{n-1}, with weights $\{W_{n-1}^i\}_{i=1}^N$, and you update their weights to $\{\tilde{W}_{n-1}^i\}_{i=1}^N$ so that these draws reflect the stage n distribution $\pi_n(\theta)$ rather than the stage $n-1$ distribution π_{n-1}.

The mutation step changes the particle values, leaving weights unchanged. In the absence of the mutation step, the particle values would be restricted to the set of values drawn in the initial stage from the proposal density $\pi_0(\theta)$. This would clearly be inefficient, because the latter is typically a poor approximation of the posterior.[16] Instead, as the algorithm cycles through the N_ϕ stages, the particle values successively adapt to the shape of the posterior distribution. This is the key difference between SMC and importance sampling. The mutation step is the only computationally expensive step in the SMC algorithm. However, it is

[16] Without changing particle values, some values would get smaller and smaller weights in the correction step and eventually disappear in the selection step, without new ones added. One would add up with only a few values for all particles, i.e. many duplicates, or even a single value duplicated N times; the value which happened to have the highest mass under the target distribution.

independent across particles and therefore can be parallelized.[17]

The selection step aims at throwing away particles with very low weights. It does so by sampling a set of particle values $\{\hat{\theta}_n^i\}_{i=1}^{N_\phi}$ from the set of values $\{\theta_{n-1}^i\}_{i=1}^N$ with weights $\{\tilde{W}_n^i\}_{i=1}^N$ (multinomial distribution). It is performed only every few stages.[18] The decision of whether or not to resample in a given stage n is typically based on a threshold rule for the variance of the particle weights:

$$\widehat{ESS}_n = N \Big/ \left(\frac{1}{N} \sum_{i=1}^{N} (\tilde{W}_n^i)^2 \right) ,$$

where ESS stands for effective sample size. If the particles have equal weights, then $\widehat{ESS}_n = N$. If one particle has weight N and all other particles have weight 0, then $\widehat{ESS}_n = 1$. To balance the trade-off between adding noise and equalizing particle weights, the resampling step is executed if $\widehat{ESS}_n$ falls below $N/2$ (for example).

As the above formula illustrates, different choices of the sequence of likelihood functions $\{p_n(Y|\theta)\}_{n=0}^{N_\phi}$ lead to different sequences of bridge distributions $\{\pi_n\}_{n=0}^{N_\phi}$, including the proposal distribution π_0. Under likelihood tempering,

$$p_n(Y|\theta) = p(Y|\theta, M_1)^{\phi_n} \ , \ , \quad \phi_0 = 0, \quad \phi_{N_\phi} = 1, \quad \phi_n \uparrow 1 \ .$$

This means that the algorithm starts out from the prior distribution as the proposal distribution – $\pi_0(\theta) = p(\theta)$ – and successively adds more weight to the likelihood until finally the posterior is reached. Under (generalized) data tempering (Cai et al., 2021),

$$p_n(Y|\theta) = p(Y_{1:T}|\theta)^{\phi_n} p(Y_{1:T_0}|\theta)^{1-\phi_n} \ , \quad T_0 < T \ ,$$

i.e. the bridge distributions in the SMC algorithm start out from the "short sample-posterior" $p(\theta|Y_{1:T_0})$ and reach the "full sample-posterior" $p(\theta|Y_{1:T})$ by gradually shifting the weight from the short sample-likelhood to the full sample-likelihood. This is useful if a model was already estimated at some point in the past (T_0) and now needs to be re-estimated using additional data that became available in the meantime ($T_1 - T_0$). Finally, under model

[17] The proposal density in the mutation step can be specified based on the (current) particle approximation of π_n, the particles $\{\theta_{n-1}^i, \tilde{W}_n^i\}_{i=1}^N$ from the correction step. For example, if RWMH is used, one can compute Σ_n as the approximation of $\mathbb{V}_{\pi_n}$ based on $\{\theta_{n-1}^i, \tilde{W}_n^i\}_{i=1}^N$. The scaling c_n can be chosen adaptively to target an acceptance ratio of 0.25 using the rule $c_n = c_{n-1} f(\text{AR}_{n-1})$, where $f(x) = 0.95 + 0.1 \frac{exp\{16(x-0.25)\}}{1+exp\{16(x-0.25)\}}$ where AR_{n-1} is the acceptance rate from the previous stage. If the acceptance rate was below (above) 25%, the scaling factor is decreased (increased).

[18] On the one hand, resampling adds noise to the approximation. On the other hand, it equalizes the particle weights, which increases the accuracy of subsequent importance sampling approximations. If resampling is not performed, then simply $\hat{\theta}_n^i = \theta_{n-1}^i$ and $W_n^i = \tilde{W}_n^i$ for each i.

tempering (Mlikota and Schorfheide, 2023),

$$p_n(Y|\theta) = p(Y|\theta, M_1)^{\phi_n} p(Y|\theta, M_0)^{1-\phi_n} \ ,$$

i.e. the bridge distributions in the SMC algorithm start out from the posterior $p(\theta|Y, M_0)$ under an approximate model M_0 and reach the posterior $p(\theta|Y, M_1)$ under the original model of interest M_1 by gradually shifting the weight from the likelihood under M_0 to the likelihood under M_1.

The sequence of tempering parameters $\{\phi_n\}_{n=0}^{N_\phi}$ can be chosen using a fixed tempering schedule as $\phi_n = (n/N_\phi)^\lambda$ for some pre-specified λ and number of stages N_ϕ.[19] A more flexible and powerful version of the SMC algorithm is obtained by choosing ϕ_n adaptively to target a desired level of the ESS.[20] Write the incremental weights $\{\tilde{w}_n^i\}_{i=1}^N$ above as

$$\tilde{w}^i(\phi) = \frac{p(Y|\theta_{n-1}^i, M_1)^\phi p(Y|\theta_{n-1}^i, M_0)^{1-\phi}}{p(Y|\theta_{n-1}^i, M_1)^{\phi_{n-1}} p(Y|\theta_{n-1}^i, M_0)^{1-\phi_{n-1}}} \ .$$

Given the incremental weights, we can compute the normalized weights and $\widehat{ESS}_n(\phi)$ also as a function of ϕ. Next, define

$$f(\phi) = \widehat{ESS}_n(\phi) - \alpha \widehat{ESS}_{n-1}^*$$

for some $\alpha \in (0, 1)$ and let $\phi_n = \min\{\phi_n^*, 1\}$, where ϕ_n^* satisfies $f(\phi_n^*) = 0$.[21] For example, $\alpha = 0.95$ chooses $\{\phi_n\}_{n=1}^{N_\phi}$ so as to let the ESS go down by 5% in every stage. The number of stages needed to reach the target distribution $\pi(\theta)$ is then determined by the algorithm itself; the researcher only specifies the proposal distribution π_0 and a rule of how to approach the target distribution π, and the algorithm determines how many stages are needed to do that for a (roughly) given approximation accuracy, determined by α.

Finally, it is important to store at each stage n not only the particles $\{\theta_n^i, W_n^i\}_{i=1}^N$, but also the corresponding values of the likelihood and prior evaluated at each θ_n^i. They are needed in the correction and mutation steps for computing the incremental weights and propagating the particle values via the MH algorithm.

[19] $\lambda = 2$ works well in practice, and it is best to have N_ϕ roughly equal to N or $N/2$. See Herbst and Schorfheide (2015).

[20] See Cai et al. (2021).

[21] Thereby, $\widehat{ESS}_{n-1}^* = \widehat{ESS}_{n-1}$ if the stage $n-1$ selection step (resampling) was executed and $\widehat{ESS}_{n-1}^* = N$ otherwise.