

Supplement to
**“Inference by Stochastic Optimization:
A Free-Lunch Bootstrap”**

Jean-Jacques Forneron*

Serena Ng[†]

April 30, 2020

This Supplemental Material consists of Appendices B and C to the main text.

*Department of Economics, Boston University, 270 Bay State Rd, MA 02215 Email: jjmf@bu.edu

[†]Department of Economics, Columbia University and NBER, 420 W. 118 St. MC 3308, New York, NY 10027 Email: serena.ng@columbia.edu

Appendix B Implementing rNR in R

To illustrate how the rNR is implemented in a real-data setting, we provide some detailed commented R code below which estimates a probit model on the Mroz (1987) data.

```
set.seed(123)      # set the seed
library(numDeriv)  # compute numerical derivatives using finite differences,
                  # alternative: library(pracma) is usually faster
library(foreign)   # to load the data set in Stata dta format

data = read.dta('mroz.dta') # read the mroz data

y = data$inlf # outcome variable
X = cbind(data$nwfeinc, data$educ, data$exper, # regressors
          data$exper^2, data$age, data$kidslt6, data$kidsge6, 1)

colnames(X) = c('nwfeinc', 'educ', 'exper', 'exper2', # labels
               'age', 'kidslt6', 'kidsge6', 'constant')

n = 753          # sample size
index0 = 1:n    # indices for the sample data

loglik <- function(coef, index=index0) {
  # compute the log-likelihood for the Probit model on the
  # observations indexed by index (default 1:n, the original sample)
  # at theta = coef

  score = X[index,] %*% coef # compute the z-scores
  ll     = y[index] * log( pnorm(score) ) +
          (1-y[index]) * log( 1-pnorm(score) )
  return( sum( ll ) )
}

d_loglik <- function(coef, index=index0) {
  # compute the gradient of the log-likelihood for the Probit model on
  # the observations indexed by index (default 1:n, the original
  # sample) at theta = coef
  # In this example, the gradient is analytically tractable, it could
  # be evaluated by finite differences by using the following:
  # d_loglik <- function(coef, index=index0) { return(jacobian(loglik,
  # coef, index=index)) }

  yy = y[index] # keep observations indexed by index
  XX = X[index,] # keep observations indexed by index
  score = XX %*% coef # compute the z-score
  dll   = 0 # initialize the gradient

  for (i in 1:length(index)) {
    dll = dll +
      (yy[i] * XX[i,] * dnorm(score[i]) / pnorm(score[i]) -
```

```

    (1-yy[i])*XX[i,]*dnorm(score[i])/(1-pnorm(score[i])))
  }
  return(dll)
}

rNR <- function(coef0, learn = 0.1, iter = 500, m = n) {
  # generate 'B = iter' rNR draws with learning rate 'gamma = learn'
  # with m out of n resampling

  coefs      = matrix(NA,iter,length(x0)) # matrix where draws will be
    stored
  coefs[1,] = coef0 # initialize the first-draw

  for (i in 2:iter) {
    index = sample(1:n,m,replace=TRUE) # sample m out of n
      observations with replacement

    G = d_loglik(coefs[i-1,],index=index) # compute the resampled
      gradient G using analytical derivatives. Alternative using
      finite differences:
    # G = jacobian(loglik,coefs[i-1,],index=index)
    H = hessian(loglik,coefs[i-1,],index=index) # compute the
      resampled hessian H using finite differences; we could also
      compute the jacobian of the gradient d_loglik

    coefs[i,] = coefs[i-1,] - learn*solve(H,G) # update
  }
  colnames(coefs) = colnames(X) # label the coefficients
  return( list(coefs = coefs) ) # return draws
}

# estimates and standard errors (source: Introductory Econometrics, A
  Modern Approach 2nd Edition, Wooldridge)
coef = c(-0.012,0.131,0.123,-0.0019,-0.053,-0.868,0.036,0.270)
ses   = c( 0.005,0.025,0.019, 0.0006, 0.008, 0.119,0.043,0.509)

iter_rNR = 2e3          # number of rNR draws
learn     = 0.3          # learning rate
coef0     = coef*3.25    # starting value

m1 = 753 # m = n
m2 = 200 # m = 200
m3 = 100 # m = 100

# adjustments to get valid standard errors
adj_rnr1 = sqrt(m1/n)*sqrt( (1-(1-learn)^2)/learn^2 )
adj_rnr2 = sqrt(m2/n)*sqrt( (1-(1-learn)^2)/learn^2 )
adj_rnr3 = sqrt(m3/n)*sqrt( (1-(1-learn)^2)/learn^2 )

b1 = 1 + round(log(0.01)/log(1-learn)) # burn-in sample size

# generate rNR draws

```

```

out_rNR1 = rNR(coef0,learn,b1 + iter_rNR, m1)
out_rNR2 = rNR(coef0,learn,b1 + iter_rNR, m2)
out_rNR3 = rNR(coef0,learn,b1 + iter_rNR, m3)

# format output
estimates =
  rbind( params,
    apply(out_rNR1$coef[b1:(iter_rNR+b1)],2,mean),
    apply(out_rNR2$coef[b1:(iter_rNR+b1)],2,mean),
    apply(out_rNR3$coef[b1:(iter_rNR+b1)],2,mean))

std_errs =
  rbind( ses,
    apply(out_rNR1$coef[b1:(iter_rNR+b1)],2,sd)*adj_rnr1,
    apply(out_rNR2$coef[b1:(iter_rNR+b1)],2,sd)*adj_rnr2,
    apply(out_rNR3$coef[b1:(iter_rNR+b1)],2,sd)*adj_rnr3)

estimates = as.data.frame(estimates)
colnames(estimates) = colnames(X)
rownames(estimates) = c('MLE','rNRn','rNR200','rNR100')

std_errs = as.data.frame(std_errs)
colnames(std_errs) = colnames(X)
rownames(std_errs) = c('ase','rNRn','rNR200','rNR100')

# print results
print(round( cbind( t(estimates), t(std_errs) ), digits = 3 ))

# output printed below:
#
#           MLE      rNRn rNR200 rNR100      ase      rNRn rNR200 rNR100
#nwifeinc -0.012 -0.012 -0.013 -0.014  0.005  0.005   0.005  0.005
#educ      0.131  0.132  0.136  0.140  0.025  0.026   0.026  0.028
#exper      0.123  0.123  0.123  0.125  0.019  0.019   0.020  0.021
#exper2    -0.002 -0.002 -0.002 -0.002  0.001  0.001   0.001  0.001
#age       -0.053 -0.053 -0.054 -0.055  0.008  0.008   0.009  0.009
#kidslt6   -0.868 -0.872 -0.895 -0.917  0.119  0.121   0.121  0.126
#kidsge6    0.036  0.038  0.040  0.038  0.043  0.045   0.047  0.049
#constant  0.270  0.272  0.282  0.276  0.509  0.506   0.505  0.535

```

Appendix C Additional Empirical and Simulation Results

C.1 Simulated Examples

Example 2: MA(1) The following provides additional details on computing the estimates found in Table 3. The data generating process is $y_t = \mu + e_t + \psi e_{t-1}$ where $e_t \sim \mathcal{N}(0, 1)$ iid. For a given $\theta = (\mu, \psi)$, the filtered residuals are computed as $e_t(\theta) = y_t - \mu - \psi e_{t-1}(\theta)$ initialized with $e_0 = 0$. The NLLS objective is then $Q_n(\theta) = \sum_{t=1}^n e_t(\theta)^2$. To find the gradient of Q_n we compute the jacobian and the Hessian of $x_t(\theta) = \mu + \psi e_{t-1}(\theta)$ which are given by:

$$\nabla x_t(\theta) = \begin{pmatrix} 1 \\ \psi \frac{de_{t-1}(\theta)}{d\psi} + e_{t-1}(\theta) \end{pmatrix}, \quad \nabla^2 x_t(\theta) = \begin{pmatrix} 0 & 0 \\ 0 & \frac{de_{t-1}(\theta)}{d\psi} \end{pmatrix}.$$

The gradient of Q_n is $G_n(\theta) = 2 \sum_{t=1}^n e_t(\hat{\theta}_n) \nabla x_t(\hat{\theta}_n) = 0$. Similarly, the Hessian is $H_n(\theta) = 2 \sum_{t=1}^n [e_t(\hat{\theta}_n) \nabla^2 x_t(\hat{\theta}_n) + \nabla x_t(\hat{\theta}_n) \nabla' x_t(\hat{\theta}_n)]$. The objective is minimized using Newton-Raphson iterations based on the analytical G_n, H_n . The asymptotic standard errors are computed from the inverse Hessian, based on the information matrix equality.

For the standard bootstrap, we implement a resampling scheme designed for State-Space models described in Stoffer and Wall (2004). Given a converged estimate $\hat{\theta}_n$, compute the filtered $e_t(\hat{\theta}_n)$. The resampled data is then generated as $y_t^{(b)} = \hat{\mu}_n + e_t^{(b)}(\hat{\theta}_n) + \hat{\psi}_n e_{t-1}^{(b)}(\hat{\theta}_n)$ where $e_t^{(b)}(\hat{\theta}_n)$ are iid draws with replacement taken from $\{\hat{e}_t(\hat{\theta}_n)\}_{t=1, \dots, n}$. The resampled NLLS objective $Q_n^{(b)}(\theta)$ is then computed and minimized as described above. This procedure is very time-consuming and is implemented in C++ using Rcpp to reduce computation time. Other methods described below are implemented using only R.

To implement DMK, given a converged estimate $\hat{\theta}_n$, filtered residuals $e_t(\hat{\theta}_n)$ and their derivatives, we sample indices $t_{1,b}, \dots, t_{n,b}$ with replacement from $\{1, \dots, n\}$ for each b and compute the resampled gradient and Hessian as $G_n^{(b)} = 2 \sum_{j=1}^n e_{t_{j,b}}(\hat{\theta}_n) \nabla x_{t_{j,b}}(\hat{\theta}_n)$ and $H_n^{(b)} = 2 \sum_{j=1}^n [e_{t_{j,b}}(\hat{\theta}_n) \nabla^2 x_{t_{j,b}}(\hat{\theta}_n) + \nabla x_{t_{j,b}}(\hat{\theta}_n) \nabla' x_{t_{j,b}}(\hat{\theta}_n)]$. We then generate the draws using one NR iteration $\theta_{\text{DMK}}^{(b)} = \hat{\theta}_n - [H_n^{(b)}(\hat{\theta}_n)]^{-1} G_n^{(b)}(\hat{\theta}_n)$.

To implement rNR with $m \leq n$, sample a block of m observations $(y_1^{(b)}, \dots, y_m^{(b)}) = (y_t, y_{t+1}, \dots, y_{t+m})$ with $1 \leq t \leq n-m+1$ and compute the filtered residuals $e_t^{(b)}(\theta_{b-1}) = y_t^{(b)} - \mu_{b-1} - \psi_{b-1} e_{t-1}^{(b)}(\theta_{b-1})$ for $t = 1, \dots, m$ where $(\mu_{b-1}, \psi_{b-1}) = \theta_{b-1}$ is the previous rNR draw. As above, the filtered residuals are initialized at $e_0 = 0$ and the rNR draws are initialized at $\theta_0 = (0, 0)$. Similarly to our implementation of DMK, we then we sample indices $t_{1,b}, \dots, t_{m,b}$ with replacement from $\{1, \dots, m\}$ and compute the resampled gradient and Hessian $G_m^{(b)}, H_m^{(b)}$, the updating equation gives the draws $\theta_b = \theta_{b-1} - \gamma [H_m^{(b)}(\theta_{b-1})]^{-1} G_m^{(b)}(\theta_{b-1})$.

Size of Confidence Intervals in the Simulated Examples The table below presents the size of confidence intervals over 1000 replications in the simulated examples of Section 5. Frequentist confidence intervals (ASE) are computed using $\hat{\theta}_n \pm 1.96\text{se}(\hat{\theta}_n)$. Bootstrap confidence intervals for the standard bootstrap (BOOT), DMK and KS are computed by taking the 2.5 and 97.5% percentiles of the draws $\theta^{(b)}$ except for the dynamic panel as discussed below. For rNR, the confidence intervals are computed by taking the 2.5 and 97.5% percentiles of $\bar{\theta}_{\text{RE}} + \sqrt{\frac{m}{n\phi(\gamma)}}(\theta_b - \bar{\theta}_{\text{RE}})$ where $\phi(\gamma) = \frac{\gamma^2}{1-(1-\gamma)^2}$.

For the dynamic panel, the standard bootstrap (BOOT), DMK and KS draws are adjusted so that confidence intervals are computed by taking the 2.5 and 97.5% percentiles of $\hat{\theta}_{n,\text{SMD}} + (\theta^{(b)} - \bar{\theta}_B)$, where $\bar{\theta}_B$ is the average bootstrap draw. Without this recentering the confidence intervals display significant size distortion, see Appendix C for a discussion of this recentering. For rNR, we take the 2.5 and 97.5% percentiles of $\bar{\theta}_{\text{RE},S} + \sqrt{\frac{m}{n\phi(\gamma)}}\theta_{b,S}^2$ where $\bar{\theta}_{\text{RE},S} = \frac{1}{B} \sum_{b=1}^B \theta_{b,S}^1$ after discarding the burn-in draws.

C.2 Empirical Examples

Application 1: Labor Force Participation The table below presents the estimates and standard errors for all methods and coefficients in the Mroz (1987) application.

Table C1: Labor Force Participation: Estimates and Standard Errors

	Estimates									
	MLE				rNR _n	rNR ₂₀₀	rNR ₁₀₀	rQN _n	rQN ₂₀₀	rQN ₁₀₀
nwifeinc	-0.012	-	-	-	-0.012	-0.013	-0.014	-0.012	-0.011	-0.012
educ	0.131	-	-	-	0.132	0.138	0.143	0.131	0.129	0.129
exper	0.123	-	-	-	0.123	0.124	0.123	0.123	0.124	0.125
exper2	-0.002	-	-	-	-0.002	-0.002	-0.002	-0.002	-0.002	-0.002
age	-0.053	-	-	-	-0.053	-0.053	-0.055	-0.052	-0.052	-0.052
kidslt6	-0.868	-	-	-	-0.874	-0.892	-0.902	-0.864	-0.855	-0.844
kidsge6	0.036	-	-	-	0.037	0.038	0.041	0.036	0.035	0.032
const.	0.270	-	-	-	0.271	0.216	0.234	0.248	0.256	0.249
	Standard Errors									
	ASE	BOOT	DMK	KS	rNR _n	rNR ₂₀₀	rNR ₁₀₀	rQN _n	rQN ₂₀₀	rQN ₁₀₀
nwifeinc	0.005	0.005	0.005	0.005	0.005	0.006	0.005	0.005	0.005	0.005
educ	0.025	0.026	0.026	0.025	0.025	0.027	0.028	0.027	0.025	0.025
exper	0.019	0.020	0.019	0.019	0.019	0.020	0.021	0.019	0.018	0.017
exper2	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
age	0.008	0.009	0.008	0.008	0.009	0.008	0.009	0.009	0.008	0.008
kidslt6	0.119	0.120	0.118	0.118	0.120	0.119	0.129	0.117	0.113	0.117
kidsge6	0.043	0.046	0.045	0.045	0.045	0.048	0.047	0.044	0.042	0.045
const.	0.509	0.512	0.507	0.505	0.494	0.535	0.544	0.544	0.494	0.506

We also implemented rGD in this application to evaluate its feasibility in a real-data setting. We found that rGD requires a burn-in greater than 1000 draws to converge and the high persistence of the draws results in a very small effective sample size while the rNR converges quickly (≤ 20 draws) and has good mixing properties. This is mainly due to the ill-conditioning of the problem since $\frac{\lambda_{\min}(H_n)}{\lambda_{\max}(H_n)}$ evaluated at $\theta = \hat{\theta}_n$ is 10^{-7} which implies a very slow convergence for GD SGD and rGD.

Application 2: Earnings Dynamics During the initial convergence phase some adjustments to the rNR updating equations were required to handle the non-convexity of the objective in the Moffitt and Zhang (2018) application. For rNR and rQN, draws such that the sample objective Q_n increases 6-folds or more are discarded, i.e. we only keep θ_b if $Q_n(\theta_b) \leq 6Q_n(\theta_{b-1})$. This never occurs for rNR and rNR_w. It occurred twice for rQN and four times for rQN_w but only in the burn-in sample with BURN = 50. When a draw is discarded, the BFGS approximation of the Hessian is reset to the Hessian computed using finite differences. These adjustments ensured that rQN converged from the original starting values. For rNR_w, we reweight the observations using exponential $\mathcal{E}(1)$ draws. For KS, the score is reweighted using draws from the Rademacher distribution. Results presented in Table 5 were computed using 4 cluster nodes with an eight-core 2.6 GHz Intel Xeon E5-2650v2 processor.

Application 3: Demand for Cereal To side-step possible identification issues, we omit the income²*price interaction as well as the child*price and age-related coefficients. The results are broadly similar when the child*price coefficient is included.

The R package BLPESTIMATOR does not offer a bootstrap option. The ‘parametric’ bootstrap implemented in the Python PYBLP package of Conlon and Gortmaker (2019) draws from the asymptotic distribution, making Gaussian draws centered at $\hat{\theta}_n$ with a sandwich variance-covariance matrix. BLPESTIMATOR, implements estimation taking as input a dataset, initial values and a model specification. To implement the standard bootstrap using this package, we simply update the data by resampling at the market level and use the built-in functions to re-estimate using as initial value the sample estimate $\hat{\theta}_n$. For DMK, rNR and rQN the data is updated as described above, then built-in functions provide analytical gradient estimates. The Hessian is computed for DMK and rNR using finite differences.

Table B2 replicates the rNR estimates and standard errors from Section 5 with different learning rates $\gamma = 0.1, 0.3, 0.6$. The results are similar using $\gamma \in [0.1, 0.3]$ while $\gamma = 0.6$ is less stable and results in large standard errors for the income*price interaction coefficient.

Results presented in Table 5 were computed using 4 cluster nodes with a fourteen-core 2.4

Table C2: Demand for Cereal: Estimates and Standard Errors for $\gamma = 0.1, 0.3, 0.6$

		Estimates				Standard Errors			
		$\hat{\theta}_n$	rNR _{0.1}	rNR _{0.3}	rNR _{0.6}	BOOT	rNR _{0.1}	rNR _{0.3}	rNR _{0.6}
stdev	const.	0.284	0.264	0.266	0.260	0.129	0.126	0.127	0.176
	price	2.032	2.191	2.183	2.162	1.198	0.930	1.013	1.689
	sugar	-0.008	-0.006	-0.006	-0.005	0.017	0.011	0.011	0.017
	mushy	-0.077	-0.057	-0.056	-0.057	0.177	0.151	0.163	0.233
income	const.	3.581	3.475	3.463	3.459	0.666	0.721	0.747	1.451
	price	0.467	1.235	1.360	1.255	3.829	3.744	4.187	16.458
	sugar	-0.172	-0.170	-0.170	-0.166	0.028	0.029	0.028	0.135
	mushy	0.690	0.643	0.634	0.535	0.345	0.353	0.355	1.582

GHz Intel Xeon E5-2680v4 processor. The CH estimates were computed on a different batch job and were assigned at runtime to an eight-core 2.6 GHz Intel Xeon E5-2670 processor.

Appendix D SMD Estimation

Table D1: Dynamic Panel: Estimates of ρ and Standard Errors

S	m	Estimates				Standard Errors					
		IND	rNR _{0.3}	rNR _{0.1}	rNR _{0.01}	ASE	BOOT	DMK	rNR _{0.3}	rNR _{0.1}	rNR _{0.01}
1	200	0.619	0.589	0.592	0.590	0.045	0.049	0.050	0.034	0.034	0.025
	100	-	0.589	0.588	0.586	-	0.048	-	0.036	0.039	0.037
	50	-	0.580	0.588	0.581	-	0.050	-	0.037	0.037	0.024
2	200	0.604	0.587	0.589	0.588	0.041	0.042	0.041	0.036	0.034	0.024
	100	-	0.588	0.587	0.589	-	0.041	-	0.033	0.036	0.037
	50	-	0.588	0.592	0.581	-	0.041	-	0.037	0.038	0.038
5	200	0.578	0.589	0.590	0.590	0.037	0.038	0.037	0.033	0.035	0.034
	100	-	0.591	0.589	0.588	-	0.038	-	0.036	0.038	0.037
	50	-	0.583	0.581	0.581	-	0.038	-	0.037	0.035	0.029
10	200	0.584	0.591	0.589	0.589	0.035	0.036	0.036	0.035	0.036	0.032
	100	-	0.589	0.591	0.587	-	0.034	-	0.034	0.037	0.030
	50	-	0.586	0.589	0.587	-	0.035	-	0.038	0.031	0.032

Remark: Results reported for one simulated sample of size $n = 200, T = 5$.

SMD Estimation of a Sample Mean

To illustrate Proposition 1, consider the simple model $y_i \sim \mathcal{N}(\theta^0, 1)$. The MD estimator of θ^0 is $\hat{\theta}_{\text{MD}} = \bar{y}_n \equiv \hat{\psi}_n$. For any given θ , let $y_i^s(\theta) = \theta + e_i^s$ where $e_i^s \sim \mathcal{N}(0, 1)$. The SMD estimator is the θ that equates $\psi(\theta, y_S) = \bar{y}_{n,S}(\theta)$ to $\hat{\psi}_n$, and is found to be $\hat{\theta}_{\text{SMD}} = \hat{\theta}_{\text{MD}} - \bar{e}_{n,S}$. The rNR resamples and simulates the binding function to give

$$\theta_{b+1,S}^1 - \hat{\theta}_{n,\text{MD}} = (1 - \gamma)(\theta_{b,S}^1 - \hat{\theta}_{n,\text{MD}}) + \gamma(\hat{\theta}_{m,\text{MD}}^{(b)} - \hat{\theta}_{n,\text{MD}} - \bar{e}_{m,S}^{(b)}).$$

Note that resampling alone gives $\theta_{b+1} - \hat{\theta}_{n,\text{MD}} = (1 - \gamma)(\theta_b - \hat{\theta}_{n,\text{MD}}) + \gamma(\hat{\theta}_{m,\text{MD}}^{(b)} - \hat{\theta}_{n,\text{MD}})$. Taking conditional expectations, we have $\mathbb{E}^*(\theta_{b+1,S}^1) = \hat{\theta}_{n,\text{MD}} + (1 - \gamma)^{b+1}(\theta_0 - \hat{\theta}_{n,\text{MD}})$ so that $\mathbb{E}^*(\bar{\theta}_{\text{RE},S}) = \hat{\theta}_{n,\text{MD}} + O(\frac{1}{B})$, as in the OLS example. Furthermore, $\text{var}^*(\bar{\theta}_{\text{RE},S}) = O(\frac{1}{mB} + \frac{1}{mSB})$ where the first term is due to resampling $(\hat{\theta}_m^{(b)} - \hat{\theta}_{n,\text{MD}})$ and the second is due to simulation noise $(\bar{e}_{m,S}^{(b)})$. Hence for this example, $\bar{\theta}_{\text{RE}} = \hat{\theta}_{n,\text{MD}} + O_{p^*}(\frac{1}{B} + \frac{1}{\sqrt{mB}} + \frac{1}{\sqrt{mSB}})$, showing that by averaging over both the resampling and simulation noise, $\bar{\theta}_{\text{RE},S}$ is first-order equivalent to $\hat{\theta}_{n,\text{MD}}$ if $\frac{\sqrt{n}}{\min(m,B)} \rightarrow 0$ for any $S \geq 1$.

Part ii. of the Proposition involves a second sequence $\theta_{b,S}^2$ because the variance of the rNR draws are comprised of two quantities: $\text{var}^*(\hat{\theta}_m^{(b)})$ and $\text{var}^*(e_{m,S}^{(b)})$. But $\frac{m}{\phi(\gamma)} \text{var}^*(\theta_b) = m \text{var}^*(\hat{\theta}_m^{(b)}) + m \text{var}^*(e_{m,S}^{(b)}) > m \text{var}^*(\hat{\theta}_m^{(b)})$, and as a consequence $\text{var}^*(\hat{\theta}_m^{(b)})$ is larger than the

actual sampling uncertainty of $\bar{\theta}_{\text{RE},S}$. Running the second chain in parallel using the same resampled statistic $\{\bar{y}_m^{(b)}\}_{b=1,\dots,B}$ produces the AR(1) draws $\{\theta_{b,S}^2\}$. We can rewrite these draws as:

$$\theta_{b+1,S}^2 = (1 - \gamma)\theta_{b,S}^2 + \gamma(\hat{\theta}_{m,\text{MD}}^{(b)} - \hat{\theta}_{n,\text{MD}}).$$

This is an AR(1) process that targets the infeasible sampling distribution based on the intractable MD objective function. From the OLS example, we know that $\text{var}^*(\theta_{b,S}^2) = \frac{\gamma^2 + o(1)}{1 - [1 - \gamma]^2} \text{var}^*(\hat{\theta}_{m,\text{MD}}^{(b)})$ which is proportional to the desired variance. Hence, $\mathbb{V}_{\text{RE},S} = \frac{m}{\phi(\gamma)} \text{var}^*(\theta_b^2) = m \text{var}^*(\hat{\theta}_{m,\text{MD}}^{(b)})$ yields valid standard errors for $\bar{\theta}_{\text{RE},S}$.

Note also that the SMD bootstrap draws $\hat{\theta}_{m,\text{SMD}}^{(b)} = \hat{\theta}_{m,\text{MD}}^{(b)} - \bar{e}_{m,S}^b$ are centered around $\hat{\theta}_{n,\text{MD}}$ instead of $\hat{\theta}_{n,\text{SMD}}$ because the simulation noise $\bar{e}_{m,S}^b$ averages out. Since $\mathbb{E}^*(\hat{\theta}_{m,\text{SMD}}^{(b)}) = \hat{\theta}_{n,\text{MD}}$, the bootstrap confidence interval must be re-centered around $\hat{\theta}_{n,\text{SMD}}$ to have correct size. In contrast with rNR, the variance does not need to be adjusted. In the numerical examples below, the draws were re-centered around $\hat{\theta}_{n,\text{SMD}}$. A numerical illustration of this example is given below.

Example 4: Sample Mean To illustrate that the rNR draws achieve the same efficiency as an SMD estimators with $S = \infty$ at a lower computation cost of $S = 1$, we simulate $y_i \sim \mathcal{N}(\theta, 1)$ with $\theta = 1$, $n = 1000$. Table C2 illustrates the variance properties of rNR relative to indirect inference and the size of confidence intervals derived from the quantiles of the draws $\theta_{b,S}^2$. With $m = 200 < n = 1000$, the variance of rNR is comparable to the method of moments (which has no simulation noise) and indirect inference with $S = 20$ simulated samples of $n = 1000$ observations. The size of m out of n bootstrap confidence intervals are reported in the last line of the table for each estimator. Size for rNR is again comparable to the method of moments and indirect inference.

Table D2: Mean Estimator: standard deviation and size

	MM	rNR	IND ₁	IND ₅	IND ₁₀	IND ₂₀
std	0.031	0.032	0.047	0.035	0.033	0.032
size	0.059	0.056	0.059	0.059	0.044	0.050

Legend: $n = 1000$; rNR $\gamma = 0.3, m = 200, B = 1000$;

IND_S: indirect inference with $S = 1, 5, 10, 20$. 1000 replications.