Supporting Information. Ogle, K. and J.J. Barber. 2020. Ensuring identifiability in hierarchical mixed effects Bayesian models. *Ecological Applications.*

Appendix S1

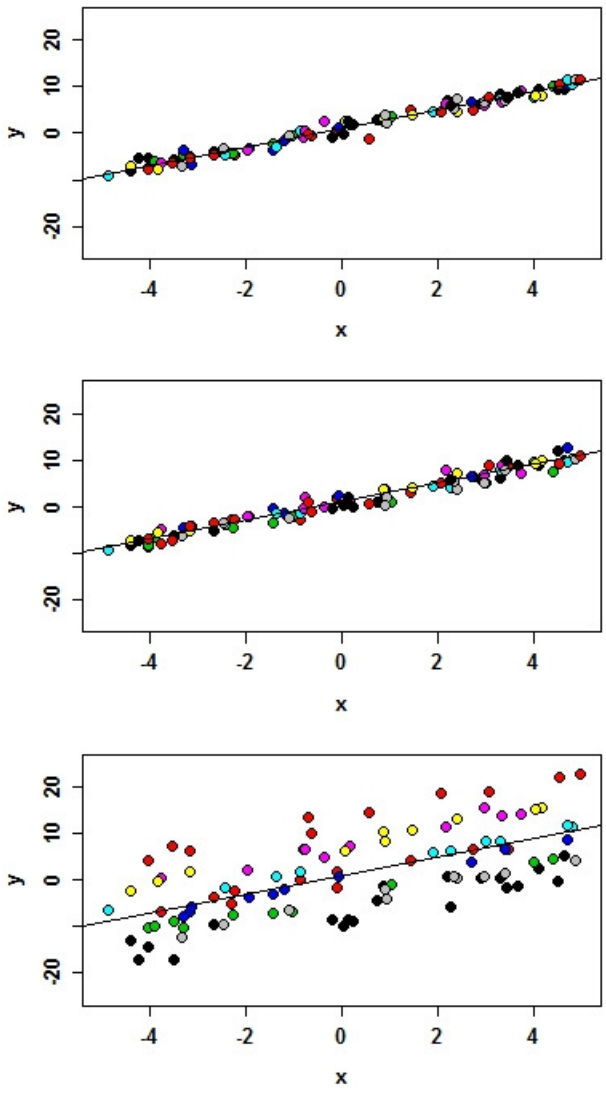


Figure S1. Synthetic data used to illustrate the random effects regression, producing results in Figures 2, 3, and 4. Data were simulated for $J = 10$ random effect levels, with 10 observations each, for a total of $N = 100$. That is, for $y_i \sim Normal(\beta_0 + \beta_1 x_i + \varepsilon_{j(i)}, \sigma^2)$ and $\varepsilon_j \sim Normal(0, \sigma_\varepsilon^2)$, data were generated from true values of $\beta_0 = 1$, $\beta_1 = 2$, $\sigma = 1$, with $x_i \sim Uniform(-5,5)$, and for **(A)** $\sigma_\varepsilon = 0.1$ (top panel), **(B)** $\sigma_\varepsilon = 1$ (middle panel), and **(C)** $\sigma_\varepsilon = 10$ (bottom panel). Points are colored by random effect level.

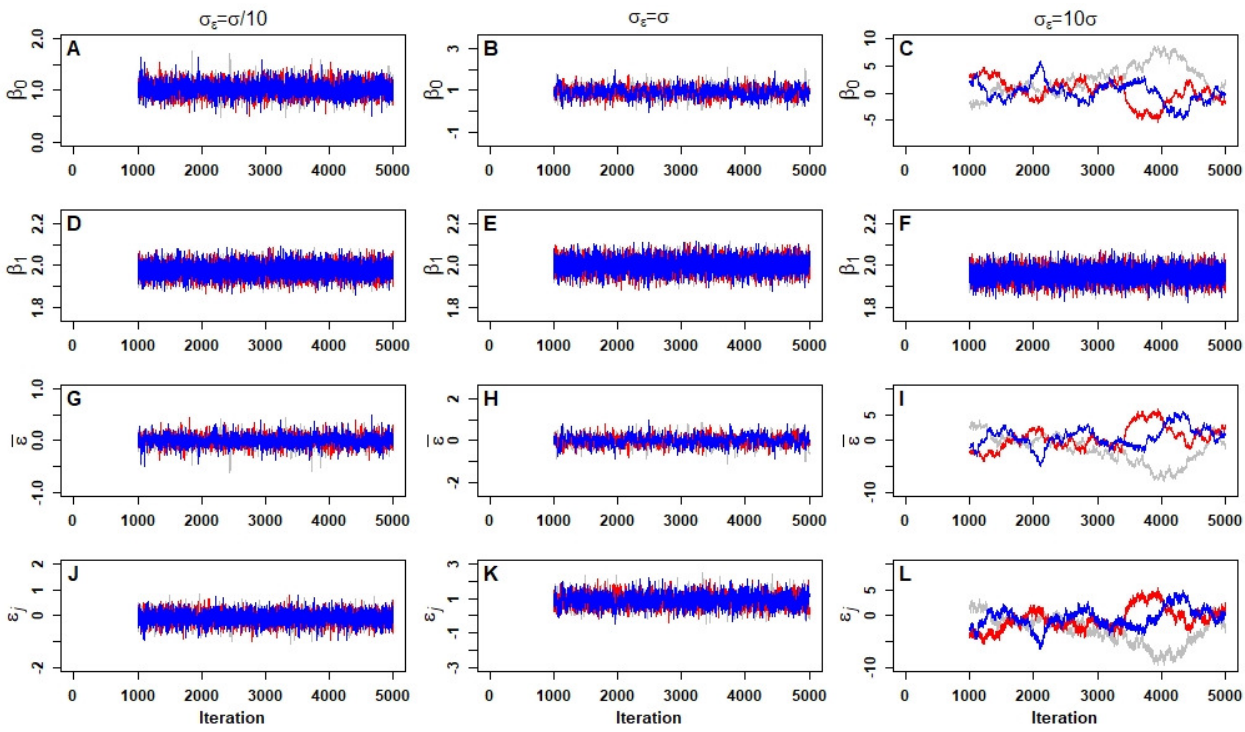


Figure S2. History plots of the MCMC samples for parameters associated with the model in Equation (5), but implemented in OpenBUGS. See Figure 3 in the main manuscript for a full definition of the model, data, and plot details; as for Figure 3, the OpenBUGS simulations are based on the models involving *Gamma*(0.1, 0.1) priors for σ and σ_ε. Here, the history plots start at iteration 1001 given that the adapting period was implemented for the first 1000 iterations.

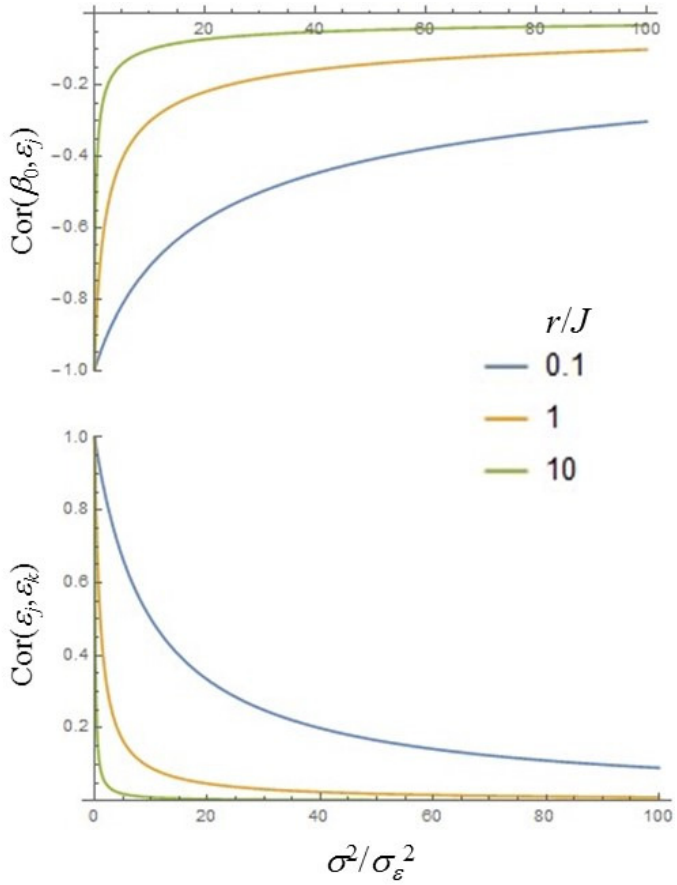


Figure S3. Correlation between the intercept (β_0) and a random effect (ε_j) (top) and between two random effects (ε_j, ε_k), based on a simple model with $\mu_i = \beta_0 + \varepsilon_j$, and with a flat prior on β_0. The correlation function is based on analytical solutions derived by Gelfand et al. (1995), as given in Gilks and Roberts (1996). For simplicity, the correlations are shown for a balanced design whereby the number of replicates (r) from each level of the random effects factor is the same for all levels, and there are J levels of the factor, for a total sample size of $N = rJ$. The correlations decrease as the ratio of the observation variance (σ^2) to the random effects variance (σ_ε^2) increases (σ_ε^2 becomes small relative to σ^2, or as $\sigma^2/\sigma_\varepsilon^2$ increases). Different colored lines correspond to different ratios of r to J.

References

Gelfand, A. E., Sahu, S. K. and Carlin, B. P. 1995. Efficient parametrizations for normal linear mixed models. - Biometrika 82: 479-488.

Gilks, W. R. and Roberts, G. O. 1996. Strategies for improving MCMC. - In: Gilks, W. R., Richardson, S. and Spiegelhalter, D. (eds.), Markov Chain Monte Carlo in Practice. Chapman & Hall/CRC.

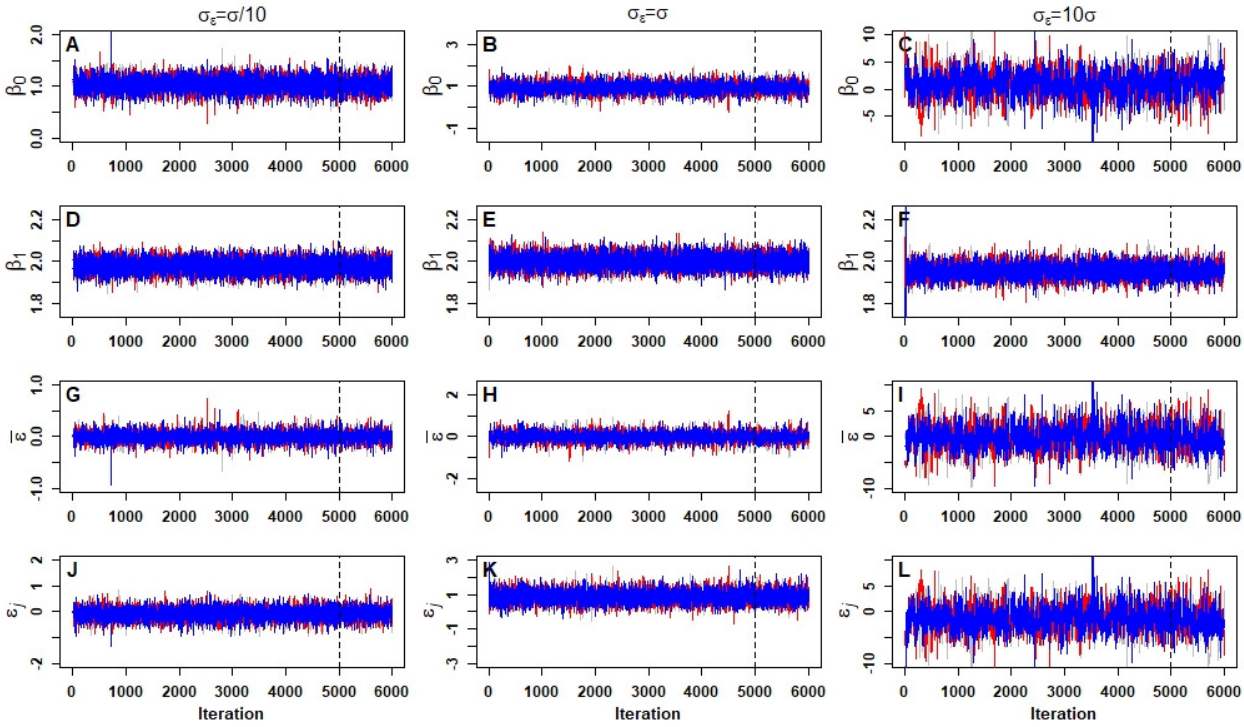


Figure S4. History plots of the posterior samples for parameters associated with the model in Equation (5), but implemented in Stan. See Figure 3 in the main manuscript for a full definition of the model, data, and plot details; as for Figure 3, the Stan simulations are based on the models involving *Gamma*(0.1, 0.1) priors for σ^{-2} and σ_ε^{-2}. Here, the history plots cover iterations 1-6000, where iterations 1-5000 denote the adapting phase (to the left of the vertical dashed line).

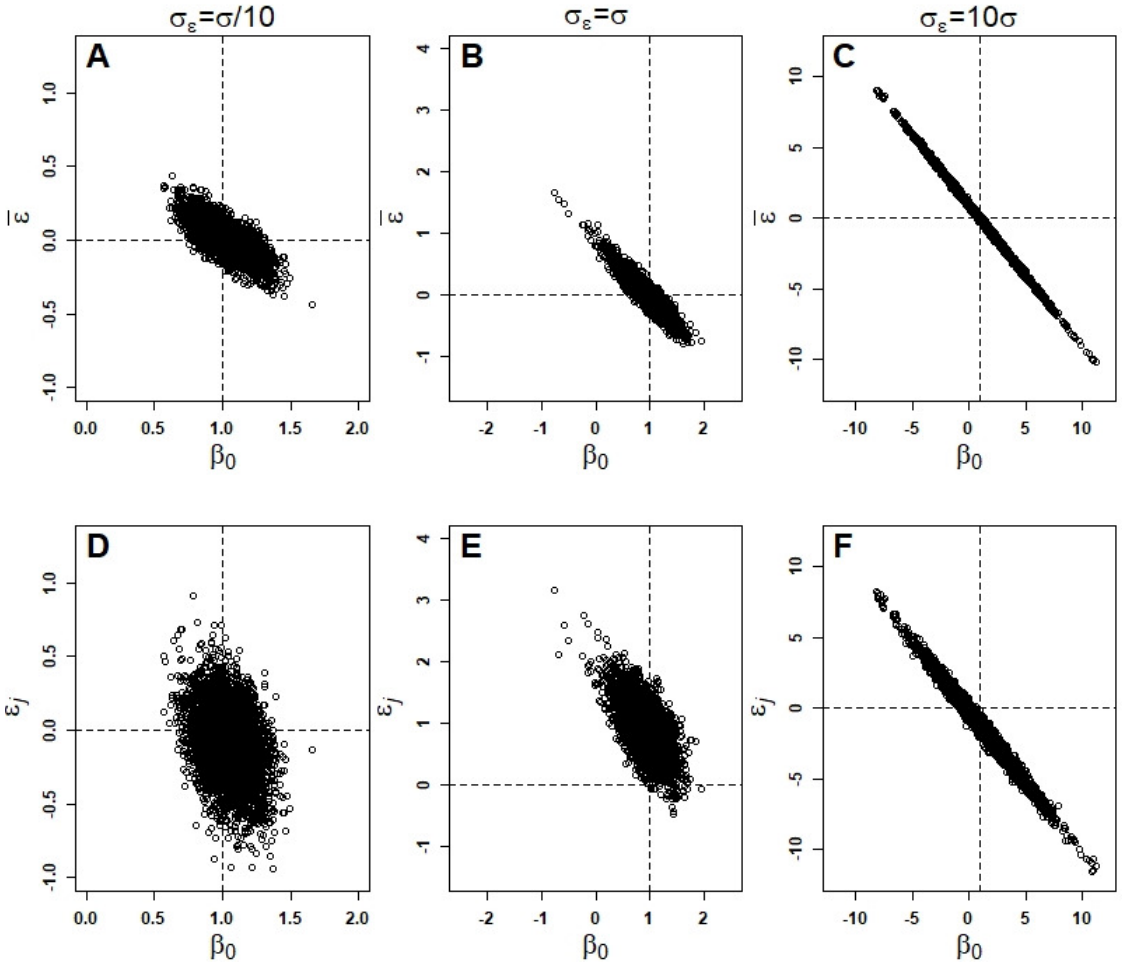


Figure S5. Bivariate scatterplots of the posterior samples obtained from the Stan simulations, after the adapting phase (iterations 5000-1000, one chain), are shown for the random effects mean ($\bar{\varepsilon}$) versus the intercept (β_0) (top row; A, B, and C) and for an individual random effect (ε_j, for $j = 4$) versus β_0 (bottom row; D, E, and F). See Figure 4 in the main text for additional details; the Stan implementation used *Gamma*(0.1, 0.1) priors for σ^{-2} and σ_ε^{-2}, just as in Figure 4. Note that despite the excellent mixing and convergence behavior associated with all three σ_ε scenarios (see Figure S5), the correlation between $\bar{\varepsilon}$ (or ε_j) and β_0 is nearly identical to that produced by the Jags (and OpenBUGS) simulations (compare against Figure 4), where the correlations become stronger as σ_ε increases relative to the observation variance (std. dev., $\sigma = 1$) such that for $\sigma_\varepsilon = 0.1$, **(A)** r = -0.71 and **(D)** r = -0.32; for $\sigma_\varepsilon = 1$, **(B)** r = -0.92 and **(E)** r = -0.60; and, for **(C)** $\sigma_\varepsilon = 10$, r = -1.00 and **(F)** r = -0.98 (F).

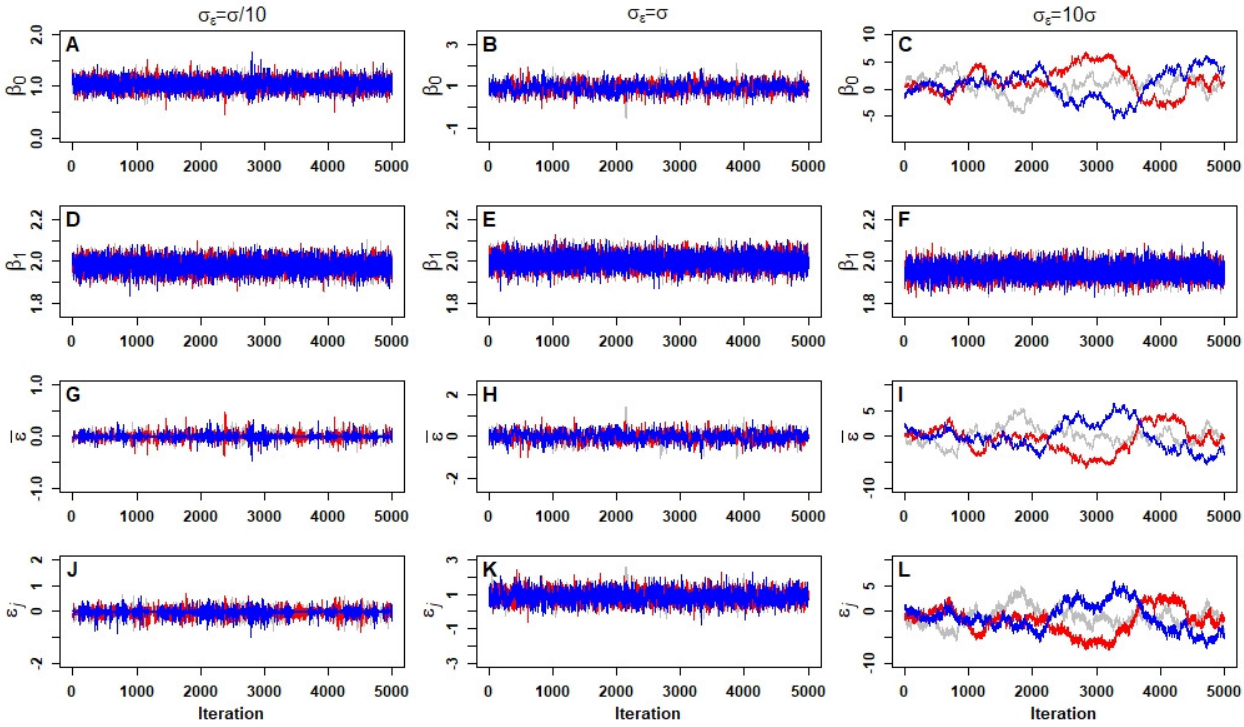


Figure S6. History plots of the MCMC samples for parameters associated with the model in Equation (5); see Figure 3 in the main manuscript for further details. In this example, a folded *Cauchy*(0,1) prior was specified for the random effects standard deviation, σ_ε. As in Figure 3, the simulated data were generated from true values of $\beta_0 = 1$, $\beta_1 = 2$, $\sigma = 1$, and for $\sigma_\varepsilon = 0.1$ (left column), $\sigma_\varepsilon = 1$ (middle column), and $\sigma_\varepsilon = 10$ (right column). The history plots correspond to the overall intercept **(A-C)** β_0, **(D-F)** slope coefficient β_1), **(G-I)** the mean of the random effects, $\bar{\varepsilon}$, and **(J-L)** an individual random effect (ε_j, for $j = 4$). All quantities show excellent mixing and convergence for the scenarios with true $\sigma_\varepsilon = 0.1$ (left column) and $\sigma_\varepsilon = 1$ (middle column), but for large σ_ε ($\sigma_\varepsilon = 10$), $\beta_0, \bar{\varepsilon}$, and ε_j exhibit very poor mixing and lack of convergence by iteration 5000 (C, I, and L, respectively). Differences in mixing and within-chain autocorrelation lead to differences in the number of posterior samples required for inference (see Table S2). Importantly, the history plots behave nearly identical to those shown in Figure 3, which specified a relatively non-informative *Gamma*(0.01, 0.01) prior for the precision, σ_ε^{-2}, indicating that the weakly informative folded *Cauchy*(0,1) prior did not solve the non-identifiability of β_0 and $\bar{\varepsilon}$ or ε_j.

Table S1. Posterior estimates (mean and 95% credible interval [CI]) for the parameters in the random effects linear regression summarized in Equation (5), and based on synthetic data described in Figure 3. The "True value" is the value of the parameter that was used to generate the synthetic data. Red italicized CIs do not contain the true value, which only occurs for three instance of σ_ε. Results are only provided for the original model parameterization, without addressing identifiability issues, implemented in three different software packages: Jags (matches results in Table 1 of the main manuscript for Approach = Orig.), Jags with a folded *Cauchy*(0,1) prior for σ_ε, which matches the results in Table S2 for Approach = Orig.), OpenBUGS, and Stan. With the exception of Jags-FC, all other results are based on models that use *Gamma*(0.1,0.1) priors for the precisions, σ^{-2} and σ_ε^{-2}.

Parameter	True value	Software*	Random effects variance scenario			Total iterations required[&]
			$\sigma_\varepsilon = 0.1\sigma = 0.1$	$\sigma_\varepsilon = \sigma = 1$	$\sigma_\varepsilon = 10\sigma = 10$	
β_0	1	Jags	1.042 (0.779, 1.309)	0.939 (0.439, 1.432)	0.921 (-3.991, 5.801)	830,547
		Jags-FC	1.039 (0.828, 1.249)	0.941 (0.431, 1.458)	0.960 (-3.736, 5.541)	210,604
		BUGS	1.038 (0.777, 1.305)	0.938 (0.432, 1.452)	0.960 (-3.736, 5.541)	1,290,899
		Stan	1.040 (0.772, 1.305)	0.937 (0.419, 1.436)	0.875 (-3.994, 5.628)	25,731
β_1	2	Jags	1.978 (1.914, 2.042)	2.002 (1.935, 2.068)	1.955 (1.886, 2.025)	3,873
		Jags-FC	1.980 (1.916, 2.044)	2.002 (1.935, 2.068)	1.955 (1.886, 2.025)	4,028
		BUGS	1.978 (1.913, 2.040)	2.001 (1.934, 2.066)	1.955 (1.886, 2.025)	4,028
		Stan	1.978 (1.913, 2.042)	2.002 (1.936, 2.067)	1.955 (1.886, 2.025)	7,967
σ	1	Jags	0.935 (0.812, 1.079)	0.959 (0.829, 1.116)	0.995 (0.860, 1.156)	3,906
		Jags-FC	0.933 (0.812, 1.077)	0.959 (0.829, 1.114)	0.995 (0.861, 1.156)	3,865
		BUGS	0.935 (0.811, 1.085)	0.959 (0.828, 1.118)	0.995 (0.861, 1.156)	3,867
		Stan	0.935 (0.812, 1.080)	0.959 (0.829, 1.115)	1.005 (0.862, 1.157)	6,877
σ_ε	Varies (see columns)	Jags	*0.289 (0.152, 0.529)*	0.720 (0.395, 1.261)	7.410 (4.677, 12.305)	3,838
		Jags-FC	0.127 (0.005, 0.381)	0.725 (0.392, 1.264)	7.064 (4.551, 11.395)	5,123
		BUGS	*0.290 (0.152, 0.530)*	0.723 (0.399, 1.270)	7.064 (4.551, 11.395)	7,552
		Stan	*0.290 (0.155, 0.541)*	0.722 (0.398, 1.250)	7.426 (4.646, 12.267)	7,967
$\bar{\varepsilon}$	0	Jags	-0.001 (-0.199, 0.192)	-0.002 (-0.462, 0.466)	-0.024 (-4.902, 4.882)	1,016,392
		Jags-FC	0.000 (-0.110, 0.112)	-0.004 (-0.489, 0.472)	-0.063 (-4.634, 4.627)	198,036
		BUGS	0.002 (-0.189, 0.193)	-0.001 (-0.483, 0.465)	-0.063 (-4.634, 4.627)	1,239,480
		Stan	0.000 (-0.193, 0.193)	0.000 (-0.468, 0.483)	0.022 (-4.742, 4.875)	27,230

*Jags-FC = Jags implementation with folded *Cauchy*(0,1) prior for σ_ε; BUGS = OpenBUGS v3.2.3.

[&]Total iterations required is only provided for the scenario $\sigma_\varepsilon = 10\sigma = 10$, and is based on the Raftery & Lewis diagnostic's (raftery.diag function) estimate of the number of iterations needed to sufficiently sample the (marginal) posterior distribution of each parameter (e.g., considering within-chain autocorrelation of each parameter).

Table S2. Posterior estimates (mean and 95% credible interval [CI]) for the parameters in the random effects linear regression summarized in Equation (5), and based on synthetic data described in Figure 3, and using a folded *Cauchy*(0,1) prior for the random effects standard deviation term (σ_ε). The "True value" is the value of the parameter that was used to generate the synthetic data. Red italicized CIs do not contain the true value, which only occurs for one instance of σ_ε. Approach is indicated by Orig. = original without addressing identifiability issues; HC = hierarchical centering (Solution 1); SZ = sum-to-zero constraints for random effects (Solution 2); and, PS = post-sweeping of random effects (Solution 4). Results for scenario $\sigma_\varepsilon = 10\sigma = 10$ are only shown.

Parameter	True value	Approach	Random effects variance scenario: $\sigma_\varepsilon = 10\sigma = 10$		
			Posterior mean	Central 95% CI	Total iterations required*
β_0	1	Orig.	0.960	(-3.736, 5.541)	210,604
		HC	0.889	(-3.787, 5.516)	4,197
		SZ	0.896	(0.697, 1.091)	3,897
		PS	0.898	(0.703, 1.096)	3,834
β_1	2	Orig.	1.955	(1.886, 2.025)	4,028
		HC	1.955	(1.887, 2.025)	3,867
		SZ	1.955	(1.885, 2.025)	3,897
		PS	1.955	(1.885, 2.024)	4,095
σ	1	Orig.	0.995	(0.861, 1.156)	3,865
		HC	0.994	(0.860, 1.153)	4,197
		SZ	0.994	(0.859, 1.156)	3,802
		PS	0.994	(0.861, 1.153)	3,865
σ_ε	10	Orig.	7.064	(4.551, 11.395)	5,123
		HC	7.050	(4.550, 11.436)	5,254
		SZ	*5.646*	*(3.630, 9.205)*	4,913
		PS	7.170	(4.582, 11.616)	4,996
$\bar{\varepsilon}$	0	Orig.	-0.063	(-4.634, 4.627)	198,036
		HC	0.007	(-4.621, 4.681)	4,302
		SZ	0	0	N/A
		PS	0	0	N/A

* See Table S1 for definition of total iterations required.

Section S1

The simulated data used to generate Figures 3-5 and Table 1 are provided here, on Github, and the simulated data are plotted in Figure S1. The data were simulated according to the model described by Equation (5) and as summarized in the legend of Figure 3. The data are in the "Data S1.RData" object and are loaded into R using the command:

```
> load("Data S1.RData")
```

This loads four variables: x, y1, y2, and y3. Each variable is of dimension 10 rows (observation) by 10 columns (group), such that each dataset contains 10 observations per 1 of 10 group levels.

The covariate (predictor variable) data are contained in x. Three different sets of response variables were simulated based on different values set for the among group variance (σ_ε) relative to the within group variance (σ, which was set to 1, $\sigma = 1$): (1) small σ_ε ($\sigma_\varepsilon = 0.1$), y1; (2) medium σ_ε ($\sigma_\varepsilon = 1$), y2; and, (3) large σ_ε ($\sigma_\varepsilon = 10$), y3.

The data are, as contained in Data S1:

x:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	2.191	-2.321	4.384	-1.427	3.307	2.948	-3.854	-1.093	0.114	-4.030
2	4.105	1.439	-3.499	-3.166	4.811	-3.788	0.884	-3.354	3.440	-0.629
3	-4.030	-3.779	3.997	3.411	2.988	-0.357	-3.169	2.961	0.234	-3.167
4	-4.392	-0.116	1.038	4.701	2.264	-1.976	4.159	4.860	0.039	4.543
5	0.740	-0.864	-3.318	-3.291	-4.873	-0.795	0.865	0.912	4.510	4.950
6	-2.661	3.468	-2.265	-0.072	-0.888	3.341	0.058	2.409	-0.216	-3.548
7	3.293	-0.105	-1.428	-1.942	-2.435	-0.789	1.448	2.311	3.650	2.069
8	0.877	2.724	-4.039	-3.140	4.705	3.713	2.397	3.387	-3.514	3.061
9	4.611	-2.237	-3.910	-1.221	1.892	2.176	4.025	-2.459	2.258	-0.700
10	2.896	-2.686	-1.044	2.679	-1.386	0.173	-4.414	0.886	-4.243	0.574

y1:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	7.073	-4.138	10.035	-3.441	7.372	5.871	-7.618	-0.623	2.602	-7.748
2	9.372	5.118	-6.118	-4.941	10.299	-6.373	3.407	-7.065	7.576	-0.596
3	-5.387	-6.372	7.742	7.186	6.561	2.575	-5.203	6.848	1.818	-5.453
4	-8.185	0.429	3.620	10.362	6.113	-3.615	8.018	11.440	-0.122	10.608
5	2.978	0.378	-4.781	-3.508	-9.010	-0.838	3.047	2.344	9.475	11.476
6	-3.812	7.600	-4.249	1.042	0.556	6.733	2.589	7.468	-0.948	-6.475
7	8.461	0.864	-2.129	-3.248	-4.757	0.620	3.908	5.359	8.646	4.552
8	2.726	5.144	-7.289	-6.740	11.402	8.970	4.671	7.191	-5.702	7.706
9	9.470	-4.620	-5.881	-1.566	4.712	6.479	8.234	-3.356	5.966	-0.311
10	6.281	-4.622	-0.453	6.739	-2.945	1.789	-7.050	3.870	-5.291	-1.045

y2:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	5.103	-2.889	7.370	-0.392	7.712	6.968	-5.464	-2.368	1.850	-6.812
2	8.770	3.091	-6.191	-5.107	10.367	-4.795	3.885	-6.157	9.768	-1.121
3	-8.684	-8.067	9.624	9.682	4.976	-0.101	-5.146	4.955	0.110	-4.237
4	-8.178	1.962	1.069	12.597	4.136	-2.019	9.953	10.362	0.276	9.043
5	1.101	-2.729	-5.932	-4.495	-9.305	-0.506	3.724	1.922	11.848	10.803
6	-5.099	8.530	-4.359	2.179	-1.301	8.889	1.395	3.638	-0.423	-7.303
7	6.123	1.770	-3.590	-1.952	-3.848	1.882	4.064	5.371	8.936	4.981
8	1.653	6.588	-8.367	-4.294	9.660	7.054	7.152	7.870	-6.166	8.721
9	10.038	-2.664	-6.742	-1.550	4.381	7.734	9.325	-3.607	5.918	0.870

```
10      5.615   -3.364  -2.261  6.468   -1.261  1.068   -7.367  0.417   -7.306  0.470

y3:
        V1      V2      V3      V4      V5      V6      V7      V8      V9      V10
1       0.850   -5.057  4.555   -3.017  8.297   15.552  -0.188  -6.600  -8.382  4.135
2       2.619   4.112   -8.714  -6.974  11.490  0.372   8.341   -12.416 -1.718  10.134
3       -14.417 -6.901  3.951   6.717   8.199   5.013   1.771   0.708   -8.838  6.069
4       -12.870 1.872   -0.859  8.748   6.091   2.233   15.536  4.173   -10.005 22.162
5       -4.442  0.015   -10.279 -7.789  -6.524  6.535   10.281  -4.077  -0.247  22.823
6       -9.511  6.742   -7.502  0.876   1.940   13.924  6.072   0.382   -8.505  7.405
7       0.314   -1.766  -7.176  -3.889  -1.814  6.627   10.858  0.810   -1.338  18.609
8       -1.464  6.422   -10.345 -5.867  11.573  14.072  13.241  1.525   -17.126 19.085
9       5.297   -2.318  -9.910  -2.153  5.902   11.325  15.255  -9.587  -5.755  13.600
10      0.542   -3.805  -6.929  3.757   0.661   7.105   -2.466  -2.002  -17.147 14.330
```

The data were simulated in R v3.5.1 using the following code:

```
# Generate synthetic datasets associated with three different
# random effects variance terms.
set.seed(19275)
y1 = array(data = NA, dim=c(10,10))
y2 = array(data = NA, dim=c(10,10))
y3 = array(data = NA, dim=c(10,10))
x = array(data = NA, dim=c(10,10))
mu = array(data = NA, dim=c(10,10))
# Small random effects variance, sig.eps = 0.1
eps1 = rnorm(n=10,mean=0,sd=0.1)
eps1 = eps1 - mean(eps1)
# Medium random effects variance, sig.eps = 1
eps2 = rnorm(n=10,mean=0,sd=1)
eps2 = eps2 - mean(eps2)
# Large random effects variance, sig.eps = 10
eps3 = rnorm(n=10,mean=0,sd=10)
eps3 = eps3 - mean(eps3)

for(j in 1:10){
  for(r in 1:10){
    x[r,j] = runif(1,min=-5,max=5)
    # Mean with true intercept = 1 and true slope = 2:
    # mu[r,j] = 1 + 2*x[r,j] + eps[j]
    y1[r,j] = rnorm(n=1, mean = 1+2*x[r,j] + eps1[j], sd = 1)
    y2[r,j] = rnorm(n=1, mean = 1+2*x[r,j] + eps2[j], sd = 1)
    y3[r,j] = rnorm(n=1, mean = 1+2*x[r,j] + eps3[j], sd = 1)
  }
}

save(x,y1,y2,y3,file = "Data S1.RData")
```

Section S2

The model code for implementing the random effect regression summarized in Equation 5 in the main text is available here, on Github, as "Code A2.R," which includes an R script for loading and preparing data, for specifying the Bayesian models, and for implementing the models in JAGS. The code does not implement any reparameterization or computational solutions to solving the non-identifiabilty of the overall intercept and random effects terms, but it does include options for specifying a relatively non-informative *Gamma* prior or a weakly informative folded-*Cauchy* prior for the random effects variance-related parameter. The model is applied to the three datasets contained in Section S1 (via the "Data S1.RData" object posted on Github). The results from this model are illustrated in Figures 3, 4, and 5A, and in Table 1 (Approach = "Orig.").

The models were implemented in R v3.5.1, via RStudio v1.1.456, and used JAGS 4.3.0.

Code A2 contains code for:

1) Loading the data (in "Data S1.RData")
2) Specifying the random effects Bayesian regression (definition of mod.string)
3) Updating the JAGS model via jags.model for each of the 3 datasets
4) Computing the Raftery diagnostics to determine the number of MCMC iterations that are required for posterior summary statistics, and to evaluate convergence of the MCMC chains.
5) Updating the JAGS model object via coda.samples to obtain a sufficient number of samples (iterations) from the posterior
6) Visualizing the MCMC samples via mcmcplot (e.g., history plots, etc.)
7) Computing posterior summary statistics for model parameters given the converged samples.

The model, as specified in model string (mod.string) will be of most interest to readers, and should be contrasted against the models in Code A3, also on Github, which shows how to recode the model to implementing the identifiability solutions.

The random effect Bayesian regression code for implementation in JAGS or OpenBUGS (as provided in Code A2) is:

```
model{
  for(j in 1:10){
    for(r in 1:10){
      # Likelihood of (stochastic) data:
      y[r,j] ~ dnorm(mu[r,j],tau)
      # Mean model for a random effects regression:
      mu[r,j] <- b0 + b1*x[r,j] + eps[j]
    }
    # Zero-centered hierarchical prior for random effects:
    eps[j] ~ dnorm(0,tau.eps)
  }
  # Relatively non-informative priors for root nodes:
  b0 ~ dnorm(0,1E-6)
  b1 ~ dnorm(0,1E-6)
```

```
  tau ~ dgamma(0.1, 0.1)
  sig <- 1/sqrt(tau)

  # Conjugate gamma prior for precision:
  tau.eps ~ dgamma(0.1, 0.1)
  sig.eps <- 1/sqrt(tau.eps)

  # Folded Cauchy(0,A^2,df=1) prior for standard deviation:
  # alpha ~ dnorm(0,1)
  # tau.temp ~ dgamma(a,b)
  # sig.temp <- 1/sqrt(tau.temp)
  # sig.eps <- abs(alpha)*sig.temp
  # tau.eps <- pow(sig.eps,-2)

  # Compute mean of random effects (to illustrate non-identifiability)
  mean.eps <- mean(eps[])
}
```

Section S3

The model code for implementing Solutions 1 (hierarchical centering, HC), 2 (sum-to-zero, SZ), and 4 (post-sweeping, PS) with the random effect regression summarized in Equation 5 in the main text is available here, on Github, as "Code A3.R," which includes an R script for loading and preparing data, for specifying the Bayesian models, and for implementing the models in JAGS. The models are applied to the three datasets contained in Data S1. Results from these model are illustrated in Figures 5B (HC), 5C (SZ), 5D (PS), and in Table 1.

The models were implemented in R version 3.5.1, via RStudio version 1.1.456, and used JAGS version 4.3.0.

Code A3 contains code for:

1) Loading the data (in "Data S1.RData")
2) Specifying the random effects Bayesian regression using the three different solutions (definition of mod.string.hc, mod.string.sz, and mod.string.ps)
3) Initializing the JAGS models via jags.model and updating the jags.model objects using coda.samples for each of the three model implementation versions.
4) Visualizing the MCMC samples via mcmcplot (e.g., history plots, etc.)
5) Computing the Raftery diagnostics to determine the number of MCMC iterations that are required for posterior summary statistics, and to evaluate convergence of the MCMC chains.
6) Computing posterior summary statistics for model parameters given the converged samples.

The models, as specified in the model strings (mod.string.XX), will be of most interest to readers, and should be contrasted against the model in Code A2, which represents the original, non-identifiable implementation.

The code for implementing the models, for each solution, in JAGS or OpenBUGS (as provided in Code A3) is:

```
############################################################
### JAGS (OpenBUGS) code for hierarchical centering (hc).
############################################################

model{
  for(j in 1:10){
    for(r in 1:10){
      y[r,j] ~ dnorm(mu[r,j],tau)
      # Mean model differs from the original, non-identifiable model:
      mu[r,j] <- b0.eps[j] + b1*x[r,j]
    }
    # Hierarchical prior for random effects, centered on overall intercept:
    b0.eps[j] ~ dnorm(b0,tau.eps)
    # If interested in monitoring the random effects, can
    # compute as a deviation term (but, not necessary)
    eps[j] <- b0.eps[j] - b0
  }
```

```
  # Relatively non-informative priors for root nodes:
  b0 ~ dnorm(0,1E-6)
  b1 ~ dnorm(0,1E-6)
  tau ~ dgamma(0.1, 0.1)
  sig <- 1/sqrt(tau)
  # Mean of random effects (not necessary):
  mean.eps <- mean(eps[])

  # Conjugate gamma prior for random effects precision:
  tau.eps ~ dgamma(0.1, 0.1)
  sig.eps <- 1/sqrt(tau.eps)

  # Folded Cauchy(0,A^2,df=1) prior for standard deviation:
  # alpha ~ dnorm(0,1)
  # tau.temp ~ dgamma(a,b)
  # sig.temp <- 1/sqrt(tau.temp)
  # sig.eps <- abs(alpha)*sig.temp
  # tau.eps <- pow(sig.eps,-2)
}


############################################################
### JAGS (OpenBUGS) code with sum-to-zero constraints (sz).
############################################################

model{
    for(j in 1:10){
      for(r in 1:10){
        y[r,j] ~ dnorm(mu[r,j],tau)
        # Mean model the same as the original, non-identifiable model:
        mu[r,j] <- b0 + b1*x[r,j] + eps[j]
      }
    }

    # Hierarchical prior for all but one random effect:
    for(j in 1:9){
      eps[j] ~ dnorm(0,tau.eps)
    }
    # Remaining random effect = minus sum of other random effects:
    eps[10] <- -sum(eps[1:9])

    # Relatively non-informative priors for root nodes:
    b0 ~ dnorm(0,1E-6)
    b1 ~ dnorm(0,1E-6)
    tau ~ dgamma(0.1, 0.1)
    sig <- 1/sqrt(tau)
    # Mean of the random effects (not necessary), which will be exactly zero:
    mean.eps <- mean(eps[])

    # Conjugate gamma prior for random effects precision:
    tau.eps ~ dgamma(0.1, 0.1)
    sig.eps <- 1/sqrt(tau.eps)

    # Folded Cauchy(0,A^2,df=1) prior for standard deviation:
    # alpha ~ dnorm(0,1)
    # tau.temp ~ dgamma(a,b)
    # sig.temp <- 1/sqrt(tau.temp)
```

```
    # sig.eps <- abs(alpha)*sig.temp
    # tau.eps <- pow(sig.eps,-2)
  }


############################################################
### JAGS (OpenBUGS) code for post-sweeping of random
### effects (ps).
############################################################

model{
    for(j in 1:10){
      for(r in 1:10){
        y[r,j] ~ dnorm(mu[r,j],tau)
        # Mean model same as original, non-identifiable model:
        mu[r,j] <- b0 + b1*x[r,j] + eps[j]
      }
      # Zero-centered hierarchical prior for random effects, just
      # as used in the original model (do not monitor or report these):
      eps[j] ~ dnorm(0,tau.eps)
      # Compute identifiable random effects (monitor and report these,
      # if desired):
      eps.star[j] <- eps[j] - mean.eps
    }
    # Prior for non-identifiable intercept (don't monitor or report)
    b0 ~ dnorm(0,1E-6)
    # Compute identifiable intercept (monitor and report this):
    b0.star <- b0 + mean.eps
    # Relatively non-informative priors for other root nodes:
    b1 ~ dnorm(0,1E-6)
    tau ~ dgamma(0.1, 0.1)
    sig <- 1/sqrt(tau)
    # Mean of non-identifiable random effects (required)
    mean.eps <- mean(eps[])
    # Mean of identifiable random effects (not required) (will be zero).
    mean.eps.star <- mean(eps.star[])

    # Conjugate gamma prior for random effects precision:
    tau.eps ~ dgamma(0.1, 0.1)
    sig.eps <- 1/sqrt(tau.eps)

    # Folded Cauchy(0,A^2,df=1) prior for standard deviation:
    # alpha ~ dnorm(0,1)
    # tau.temp ~ dgamma(a,b)
    # sig.temp <- 1/sqrt(tau.temp)
    # sig.eps <- abs(alpha)*sig.temp
    # tau.eps <- pow(sig.eps,-2)
  }
```

Section S4

The model code for implementing Solutions 1 (hierarchical centering, HC), 2 (sum-to-zero, SZ), and 4 (post-sweeping, PS) with the nested random effects regression summarized in Equation 11 in the main text is available here, on Github, as "Code A4.R," which includes an R script for loading and preparing data, for specifying the Bayesian models, and for implementing the models in JAGS. The models are applied a pseudo dataset that is contained within the Code A4 R script file.

The models were implemented in R version 3.5.1, via RStudio version 1.1.456, and used JAGS version 4.3.0.

Code A4 contains code for:

1) Defining the data (data.eq11 as a list)
2) Specifying the nested random effects Bayesian regression model using the three different solutions (definition of mod.string.hc, mod.string.sz, and mod.string.ps)
3) Initializing the JAGS models via jags.model and updating the jags.model objects using coda.samples for each of the three model implementation versions.
4) Visualizing the MCMC samples via mcmcplot (e.g., history plots, etc.)
5) Computing the Raftery diagnostics to determine the number of MCMC iterations that are required for posterior summary statistics, and to evaluate convergence of the MCMC chains.
6) Computing posterior summary statistics for model parameters given the converged samples.

The code for implementing the models, for each solution, in JAGS or OpenBUGS (as provided in Code A4) is given below. Note that relatively non-informative *Gamma* priors are used for the random effects precision terms, but the model code can be easily modified to include other priors, such as the folded-Cauchy priors from Code A2 and Code A3.

```
############################################################
### JAGS (OpenBUGS) code for hierarchical centering (hc).
############################################################

model{
  for(i in 1:N){
    y[i] ~ dnorm(mu[i],tau)
    # Mean model differs from the original specification in Eq 11 to reflect
    # hierarchical centering:
    mu[i] <- b0.eps[plot[i],wshed[i]] + b1*x[i]
  }
  # Hierarchical priors for random effects, centered on overall intercept
  # (b0):
  for(s in 1:Nwater){
    for(p in 1:Nplots){
      # Plot within watershed effects:
      b0.eps[p,s] ~ dnorm(b0.gamma[s], tau.eps)
    }
```

```
    # Watershed effects:
    b0.gamma[s] ~ dnorm(b0, tau.gamma)
    }

  # Relatively non-informative priors for root nodes:
  b0 ~ dnorm(0,1E-6)
  b1 ~ dnorm(0,1E-6)
  tau ~ dgamma(0.1, 0.1)
  tau.eps ~ dgamma(0.1, 0.1)
  tau.gamma ~ dgamma(0.1, 0.1)
  sig <- 1/sqrt(tau)
  sig.eps <- 1/sqrt(tau.eps)
  sig.gamma <- 1/sqrt(tau.gamma)
}


############################################################
### JAGS (OpenBUGS) code with sum-to-zero constraints (sz).
############################################################

model{
  for(i in 1:N){
    y[i] ~ dnorm(mu[i],tau)
    # Mean model as per the original specification in Eq 11:
    mu[i] <- b0 + b1*x[i] + eps[plot[i], wshed[i]] + gamma[wshed[i]]
  }
  # Zero-centered priors for random effects with sum-to-zero constraint
  # Plot within watershed effects:
  for(s in 1:Nwater){
    for(p in 1:(Nplots-1)){
      eps[p,s] ~ dnorm(0, tau.eps)
    }
    # Sum-to-zero occurs for plots within each watershed:
    eps[Nplots,s] <- -sum(eps[1:(Nplots-1),s])
  }
  # Watershed random effects:
  for(s in 1:(Nwater-1)){
    gamma[s] ~ dnorm(0,tau.gamma)
  }
  # Sum-to-zero constraint:
  gamma[Nwater] <- -sum(gamma[1:(Nwater-1)])

  # Relatively non-informative priors for root nodes:
  b0 ~ dnorm(0,1E-6)
  b1 ~ dnorm(0,1E-6)
  tau ~ dgamma(0.1, 0.1)
  tau.eps ~ dgamma(0.1, 0.1)
  tau.gamma ~ dgamma(0.1, 0.1)
  sig <- 1/sqrt(tau)
  sig.eps <- 1/sqrt(tau.eps)
  sig.gamma <- 1/sqrt(tau.gamma)
}
```

```
#################################################################
### JAGS (OpenBUGS) code for post-sweeping of random effects (ps).
#################################################################

model{
    for(i in 1:N){
      y[i] ~ dnorm(mu[i],tau)
      # Mean model as per the original specification in Eq 11:
      mu[i] <- b0 + b1*x[i] + eps[plot[i], wshed[i]] + gamma[wshed[i]]
    }
    # Zero-centered priors for non-identifiable random effects
    # Plot within watershed effects:
    for(s in 1:Nwater){
      for(p in 1:Nplots){
        # Non-identifiable random effect (don't monitor):
        eps[p,s] ~ dnorm(0, tau.eps)
        # Identifiable plot random effect (monitor this):
        eps.star[p,s] <- eps[p,s] - ave.eps[s]
      }
      # Mean plot-level random effects within each watershed:
      ave.eps[s] <- mean(eps[,s])
    }
    # Watershed random effects:
    for(s in 1:Nwater){
      # Non-identifiable random effect:
      gamma[s] ~ dnorm(0,tau.gamma)
      # Identifiable random effect (monitor this):
      gamma.star[s] <- gamma[s] + ave.eps[s] - ave.gamma - ave.ave.eps
    }
    # Mean watershed random effect:
    ave.gamma <- mean(gamma[])
    # Mean overall plot random effect:
    ave.ave.eps <- mean(ave.eps[])

    # Relatively non-informative priors for root nodes:
    # Non-identifiable intercept
    b0 ~ dnorm(0,1E-6)
    # Identifiable intercept (monitor this)
    b0.star <- b0 + ave.gamma + ave.ave.eps
    b1 ~ dnorm(0,1E-6)
    tau ~ dgamma(0.1, 0.1)
    tau.eps ~ dgamma(0.1, 0.1)
    tau.gamma ~ dgamma(0.1, 0.1)
    sig <- 1/sqrt(tau)
    sig.eps <- 1/sqrt(tau.eps)
    sig.gamma <- 1/sqrt(tau.gamma)
  }
```

Section S5

The model code for implementing Solutions 2 (sum-to-zero, SZ) and 4 (post-sweeping, PS) with the Bayesian regression involving crossed random effects, as summarized in Equation 14 in the main text is available here, on Github, as "Code A5.R," which includes an R script for loading and preparing data, for specifying the Bayesian models, and for implementing the models in JAGS. The models are applied a pseudo dataset that is contained within the Code A5 R script file.

The models were implemented in R version 3.5.1, via RStudio version 1.1.456, and used JAGS version 4.3.0.

Code A5 contains code for:

1) Defining the data (data.eq14 as a list)
2) Specifying the Bayesian regression model with cross random effects using the two different solutions (definition of mod.string.sz and mod.string.ps)
3) Initializing the JAGS models via jags.model and updating the jags.model objects using coda.samples for each of the three model implementation versions.
4) Visualizing the MCMC samples via mcmcplot (e.g., history plots, etc.)
5) Computing the Raftery diagnostics to determine the number of MCMC iterations that are required for posterior summary statistics, and to evaluate convergence of the MCMC chains.
6) Computing posterior summary statistics for model parameters given the converged samples.

The code for implementing the models, for each solution, in JAGS or OpenBUGS (as provided in Code A5) is given below. Note that relatively non-informative *Gamma* priors are used for the random effects precision terms, but the model code can be easily modified to include other priors, such as the folded-Cauchy priors from Code A2 and Code A3.

```
############################################################
### JAGS (OpenBUGS) code with sum-to-zero constraints (sz).
### Code uses the indexing variables of plot and date.
############################################################

model{
    for(i in 1:N){
      y[i] ~ dnorm(mu[i],tau)
      # Mean model as per the original specification in Eq 14:
      mu[i] <- b0 + b1*x[i] + eps[plot[i]] + gamma[date[i]]
    }
    # Zero-centered priors for random effects with sum-to-zero constraints
    # Plot random effects:
    for(p in 1:(Nplots-1)){
      eps[p] ~ dnorm(0, tau.eps)
    }
    # Sum-to-zero constraint for plots:
    eps[Nplots] <- -sum(eps[1:(Nplots-1)])
```

```
    # Date random effects:
    for(d in 1:(Ndates-1)){
      gamma[d] ~ dnorm(0,tau.gamma)
    }
    # Sum-to-zero constraint for dates:
    gamma[Ndates] <- -sum(gamma[1:(Ndates-1)])

    # Relatively non-informative priors for root nodes:
    b0 ~ dnorm(0,1E-6)
    b1 ~ dnorm(0,1E-6)
    tau ~ dgamma(0.1, 0.1)
    tau.eps ~ dgamma(0.1, 0.1)
    tau.gamma ~ dgamma(0.1, 0.1)
    sig <- 1/sqrt(tau)
    sig.eps <- 1/sqrt(tau.eps)
    sig.gamma <- 1/sqrt(tau.gamma)
  }


############################################################
### JAGS (OpenBUGS) code for post-sweeping of random effects.
############################################################

model{
    for(i in 1:N){
      y[i] ~ dnorm(mu[i],tau)
      # Mean model as per the original specification in Eq 11:
      mu[i] <- b0 + b1*x[i] + eps[plot[i]] + gamma[date[i]]
    }
    # Zero-centered priors for non-identifiable random effects
    # Plot random effects:
    for(p in 1:Nplots){
      # Non-identifiable random effect (don't monitor)
      eps[p] ~ dnorm(0, tau.eps)
      # Identifiable random effect (monitor this)
      eps.star[p] <- eps[p] - ave.eps
    }
    # Mean plot-level random effect:
    ave.eps <- mean(eps[])

    # Date random effects:
    for(d in 1:Ndates){
      # Non-identifiable random effect
      gamma[d] ~ dnorm(0,tau.gamma)
      # Identifiable random effect (monitor this)
      gamma.star[d] <- gamma[d] - ave.gamma
    }
    # Mean date random effect:
    ave.gamma <- mean(gamma[])

    # Relatively non-informative priors for root nodes:
    # Non-identifiable intercept
    b0 ~ dnorm(0,1E-6)
    # Identifiable intercept (monitor this)
    b0.star <- b0 + ave.gamma + ave.eps
    b1 ~ dnorm(0,1E-6)
```

```
  tau ~ dgamma(0.1, 0.1)
  tau.eps ~ dgamma(0.1, 0.1)
  tau.gamma ~ dgamma(0.1, 0.1)
  sig <- 1/sqrt(tau)
  sig.eps <- 1/sqrt(tau.eps)
  sig.gamma <- 1/sqrt(tau.gamma)
}
```

Section S6

Contents

S6 Stan Code for Implementing Original Model

S6.1 Preliminaries

Load data and create grouped data objects using the `nlme` package.

```
> load("../Data S1.RData")
> ls()

[1] "x"  "y1" "y2" "y3"

> s2.df<- cbind.data.frame(y1=as.vector(y1),
+                          y2=as.vector(y2),
+                          y3=as.vector(y3),
+                          x=as.vector(x),
+                          unit=rep(1:10,rep(10,10)))
> s2y1.gD<- nlme::groupedData(y1 ~ x | unit,
+                             data=s2.df,
+                             labels=list(x="x", y1="y1"))
> s2y2.gD<- nlme::groupedData(y2 ~ x | unit,
+                             data=s2.df,
+                             labels=list(x="x", y2="y2"))
> s2y3.gD<- nlme::groupedData(y3 ~ x | unit,
+                             data=s2.df,
+                             labels=list(x="x", y3="y3"))
```

S6.2 Stan Model Code

```
> writeLines(readLines("S2.v2.stan"))

data {
  // Dimensions
  int<lower=1> m;     // # of units (i.e., # groups = 10)
  int<lower=1> ni[m]; // # obs in each group (ni=10, all i)
  int<lower=1> N;     // total # of obs (sum(ni) = 100)
  int<lower=1> p;     // # of fixed effects: intercept and slope (p=2)
  int<lower=1> q;     // # of random effects per unit: intercept (q=1)

  // Data
  vector[N] y;   // observed response values
  matrix[N,p] X; // fixed effects regression matrix (1's augmented
 // with covariate (p=2))
  matrix[N,q] Z; // random effects regression matrix (q=1 column of
 // 1's)

  // Fixed effects prior hyper-parameters
  vector[p] mu_b;
  cov_matrix[p] Sig_b;
```

```
  // Random effects prior hyper-parameters
  vector<lower=0>[q] tau_eps_a;
  vector<lower=0>[q] tau_eps_b;
  vector[q] mu_eps;

  // Error precision prior hyper-parameters
  real<lower=0> tau_a;
  real<lower=0> tau_b;
}

parameters {
  vector[p] b;                // fixed effects b0 and b1 (p=2)
  matrix[q,m] eps;            // random effects epsj j=1 to m=10 (q=1)
  vector<lower=0>[q] tau_eps; // random effects precision
  real<lower=0> tau;          // obs error precision

}

transformed parameters {
  real<lower=0> sig = pow(sqrt(tau),-1);             // obs error sd
  vector<lower=0>[q] sig_eps = 1.0 ./ sqrt(tau_eps); // ran. eff. sd
}

model{
  int pos = 1;
  matrix[q,q] Sig_eps = diag_matrix(square(sig_eps)); // ran. eff. variance
  vector[N] mu_y = X*b; // fixed effects contribution to mean

  b ~ multi_normal(mu_b, Sig_b);         // fixed effect prior
  tau_eps ~ gamma(tau_eps_a, tau_eps_b); // random effects precision
 // prior
  tau ~ gamma(tau_a, tau_b);             // obs error precision prior
  for(i in 1:m) {
    eps[,i] ~ multi_normal(mu_eps, Sig_eps); // random effects
    segment(y,pos,ni[i]) ~ normal(segment(mu_y,pos,ni[i]) +
  Z[pos:(pos+ni[i]-1),]*eps[,i],
  sig);
    pos += ni[i];
  }
}
```

S6.3 Translate Stan Code to C++ with `stanc`

```
> library(rstan, quietly=TRUE)

Registered S3 methods overwritten by 'ggplot2':
method         from
```

```
[.quosures    rlang
c.quosures    rlang
print.quosures rlang
rstan (Version 2.18.2, GitRev:  2e1f913d3ca3)
For execution on a local, multicore CPU with excess RAM we recommend calling
options(mc.cores = parallel::detectCores()).
To avoid recompilation of unchanged Stan programs, we recommend calling
rstan_options(auto_write = TRUE)
```

```
> options(mc.cores = parallel::detectCores())
> rstan_options(auto_write = TRUE)
> S2.v2.stanc<- stanc(file="S2.v2.stan")
```

S6.4 Make an Executable Stan Model with `stan_model`

```
> S2.v2.stanmod<- stan_model(stanc_ret=S2.v2.stanc)
```

S6.5 Preparing Stan Data Common to All S2 Scenarios

Some minor functionality of the **nlme** package is used to prepare data for Stan; see Stan code data block.

```
> ## y ## to come later, with particular scenarios
> X<- model.matrix(y1 ~ x, random = ~1 | unit, data=s2y1.gD)
> Z<- model.matrix(nlme::reStruct(list(unit=~1), data=s2y1.gD),
+                  data=s2y1.gD)
> (N<- dim(X)[1])

[1] 100

> (p<- dim(X)[2])

[1] 2

> (ni<- table(as.numeric(as.character((s2y1.gD$unit)))))


 1  2  3  4  5  6  7  8  9 10
10 10 10 10 10 10 10 10 10 10

> (m<- length(ni))

[1] 10

> sum(ni) == N ## check

[1] TRUE
```

```
> (q<- dim(Z)[2])

[1] 1

> mu_b<- rep(0,p);
> Sig_b<- 10e6 * diag(p)
>
> tau_eps_a <- tau_eps_b <- rep(0.1, q)
> mu_eps <- rep(0, q)
>
> tau_a<- tau_b<- 0.1
>
> nchains<- 3
```

S6.6 S2 Scenario 1 sig.eps = 0.1sig

Prepare Data List to Pass to Stan

```
> S2.v2.1.stan.dat<- list(
+     m = m,
+     N = N,
+     ni = ni,
+     p = p,
+     q = q,
+     y = s2y1.gD$y1, ## sig.eps = 0.1 * sig
+     X = X,
+     Z = Z,
+     mu_b = mu_b,
+     Sig_b = Sig_b,
+     tau_eps_a = as.array(tau_eps_a,1),
+     tau_eps_b = as.array(tau_eps_b,1),
+     mu_eps = as.array(mu_eps,1),
+     tau_a = tau_a,
+     tau_b = tau_b)
```

Prepare Initial Values List to Pass to Stan

Use the **nlme** package to fit the model using restricted/residual maximum likelihood (REML). Results are used to generate initial values for Stan.

```
> s2y1.mod<- nlme::lme(y1 ~ x,
+                      random = ~1 | unit,
+                      data=s2y1.gD)
```

```
> set.seed(20500 + 5150 + 24601 + 1)
>
> ## b[p]
> b<- mvtnorm::rmvnorm(n=nchains,
+                      nlme::fixef(s2y1.mod),
+                      3*vcov(s2y1.mod))
>
> ## eps[q,m]
> mu_eps<- t(nlme::ranef(s2y1.mod))
> eps<- list(matrix(NA,q,m),matrix(NA,q,m),matrix(NA,q,m))
> Sig_eps<- unclass(nlme::getVarCov(s2y1.mod, type="random.effects"))
> for(c in 1:nchains) {
+     for(i in 1:m) {
+         eps[[c]][,i]<- mvtnorm::rmvnorm(n=q,mu_eps[,i],3*Sig_eps)
+     }
+ }
>
> ## tau_eps[q] = 1/sig.eps^2
> tau_eps<- 1000
>
> ## tau
> tau<- 1.0
>
> S2.v2.1.stan.inits<- list(
+     list(b=b[1,],
+          eps=eps[[1]],
+          tau_eps=as.array(tau_eps,1),
+          tau=tau),
+     list(b=b[2,],
+          eps=eps[[2]],
+          tau_eps=as.array(tau_eps,1),
+          tau=tau),
+     list(b=b[3,],
+          eps=eps[[3]],
+          tau_eps=as.array(tau_eps,1),
+          tau=tau))
```

Run (R) Stan

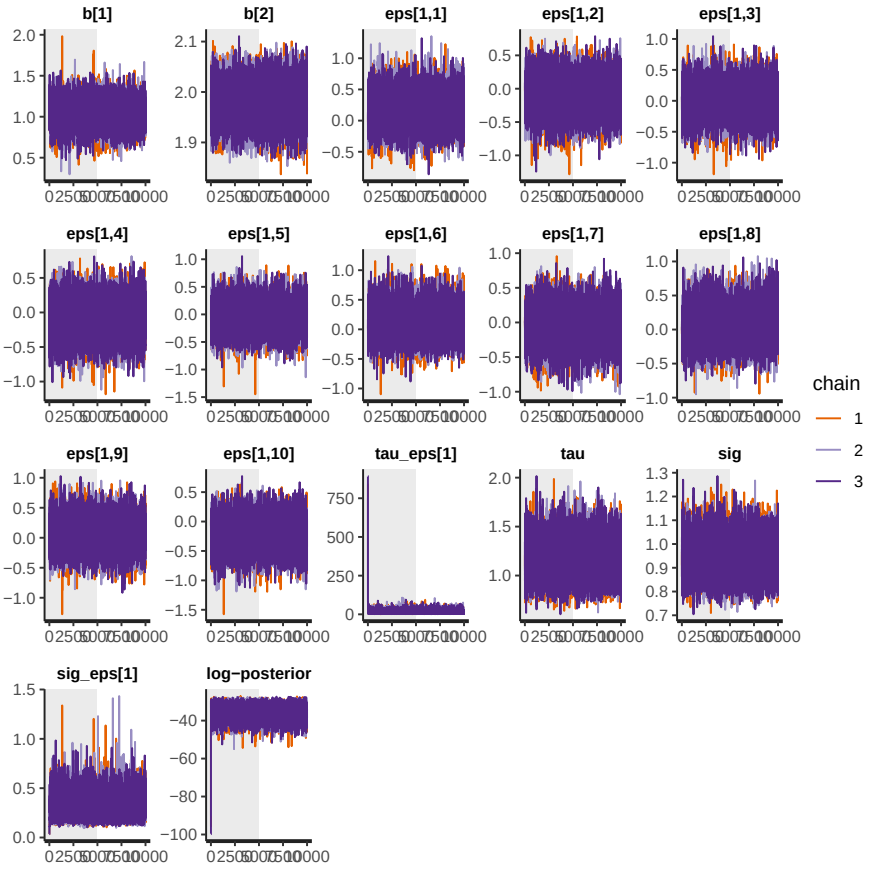
```
> S2.v2.1.stanfit<- sampling(S2.v2.stanmod,
+                            data=S2.v2.1.stan.dat,
+                            init=S2.v2.1.stan.inits,
+                            chains=nchains,
+                            iter=10000,
+                            warmup=5000,
+                            seed=c(20500+5150+24601+1),
+                            refresh=1000,
+                            sample_file="S2.v2.1.chain")
```

Summarize Posterior

```
> summary(S2.v2.1.stanfit)$summary

                   mean      se_mean          sd        2.5%          25%
b[1]          1.038081231 0.0014498703 0.13521884   0.7696439   0.95029534
b[2]          1.978234046 0.0002372099 0.03303556   1.9124787   1.95643211
eps[1,1]      0.165188768 0.0018836022 0.21995835  -0.2389946   0.01790401
eps[1,2]     -0.081153330 0.0017713926 0.21780632  -0.5311025  -0.21868321
eps[1,3]      0.004273097 0.0017812693 0.21803442  -0.4293623  -0.13441410
eps[1,4]     -0.076698962 0.0018160955 0.21437473  -0.5128522  -0.21114118
eps[1,5]     -0.028071961 0.0017483896 0.21848831  -0.4689543  -0.16691426
eps[1,6]      0.126681178 0.0018008350 0.22019341  -0.2855620  -0.02437033
eps[1,7]     -0.049590408 0.0018163383 0.21570668  -0.4891513  -0.18611863
eps[1,8]      0.077236251 0.0018075182 0.22034112  -0.3489801  -0.06636939
eps[1,9]      0.057779462 0.0017633280 0.21611477  -0.3614956  -0.08375199
eps[1,10]    -0.180429938 0.0018736712 0.22181775  -0.6468340  -0.31928274
tau_eps[1]  15.798808930 0.1094900527 9.94261380   3.5170913   8.67227255
tau          1.161989009 0.0012836478 0.16978211   0.8526029   1.04275481
sig          0.935170530 0.0005271226 0.06896106   0.8134181   0.88633173
sig_eps[1]   0.291076854 0.0011924969 0.10005800   0.1555241   0.22125590
lp__       -35.102837256 0.0500613128 3.30459389 -42.5878445 -37.10862980
                    50%          75%       97.5%     n_eff      Rhat
b[1]          1.037080182   1.12688512   1.3029605  8697.934 1.0003377
b[2]          1.978765787   2.00023657   2.0434722 19395.356 0.9998717
eps[1,1]      0.155013521   0.30150655   0.6332306 13636.488 0.9999469
eps[1,2]     -0.076592746   0.06237959   0.3344560 15118.590 0.9999006
eps[1,3]      0.005809917   0.14479450   0.4347895 14982.730 1.0001126
eps[1,4]     -0.071551534   0.06335156   0.3328061 13933.808 1.0001472
eps[1,5]     -0.027660314   0.11184128   0.4053941 15616.366 1.0000276
eps[1,6]      0.119779079   0.26666424   0.5873082 14950.674 0.9999919
eps[1,7]     -0.045864265   0.09002633   0.3712877 14103.720 1.0001868
eps[1,8]      0.071268273   0.21677810   0.5259307 14860.238 0.9998630
eps[1,9]      0.053399999   0.19414632   0.4981402 15021.134 0.9999844
eps[1,10]    -0.171235299  -0.03107500   0.2346060 14015.410 0.9998563
tau_eps[1]  13.508324996  20.42726710  41.3432434  8246.161 1.0004027
tau          1.153463571   1.27293847   1.5113755 17494.120 0.9999462
sig          0.931103713   0.97928454   1.0829954 17115.301 0.9999297
sig_eps[1]   0.272081648   0.33957330   0.5332221  7040.267 1.0004835
lp__       -34.751915472 -32.71485589 -29.7527892  4357.443 1.0006804
```

```
> traceplot(S2.v2.1.stanfit,
+           pars=names(S2.v2.1.stanfit),
+           inc_warmup=TRUE)
```



S6.7 S2 Scenario 2 sig.eps = sig

Prepare Data List to Pass to Stan

```
> S2.v2.2.stan.dat<- S2.v2.1.stan.dat
> S2.v2.2.stan.dat$y<- s2y2.gD$y2 ## sig.eps = 1 * sig
```

Prepare Initial Values List to Pass to Stan

Use the `nlme` package to fit the model using restricted/residual maximum likelihood (REML). Results are used to generate initial values for Stan.

```
> s2y2.mod<- nlme::lme(y2 ~ x,
+                      random = ~1 | unit,
+                      data=s2y2.gD)
```

```
> set.seed(20500 + 5150 + 24601 + 2)
>
> ## b[p]
> b<- mvtnorm::rmvnorm(n=nchains,
+                      nlme::fixef(s2y2.mod),
+                      3*vcov(s2y2.mod))
>
> ## eps[q,m]
> mu_eps<- t(nlme::ranef(s2y2.mod))
> eps<- list(matrix(NA,q,m),matrix(NA,q,m),matrix(NA,q,m))
> Sig_eps<- unclass(nlme::getVarCov(s2y2.mod, type="random.effects"))
> for(c in 1:nchains) {
+     for(i in 1:m) {
+         eps[[c]][,i]<- mvtnorm::rmvnorm(n=q,mu_eps[,i],3*Sig_eps)
+     }
+ }
>
> ## tau_eps[q] = 1/sig.eps^2
> tau_eps<- 1.0
>
> ## tau
> tau<- 1.0
>
> S2.v2.2.stan.inits<- list(
+     list(b=b[1,],
+          eps=eps[[1]],
+          tau_eps=as.array(tau_eps,1),
+          tau=tau),
+     list(b=b[2,],
+          eps=eps[[2]],
+          tau_eps=as.array(tau_eps,1),
+          tau=tau),
+     list(b=b[3,],
+          eps=eps[[3]],
+          tau_eps=as.array(tau_eps,1),
+          tau=tau))
```

Run (R) Stan

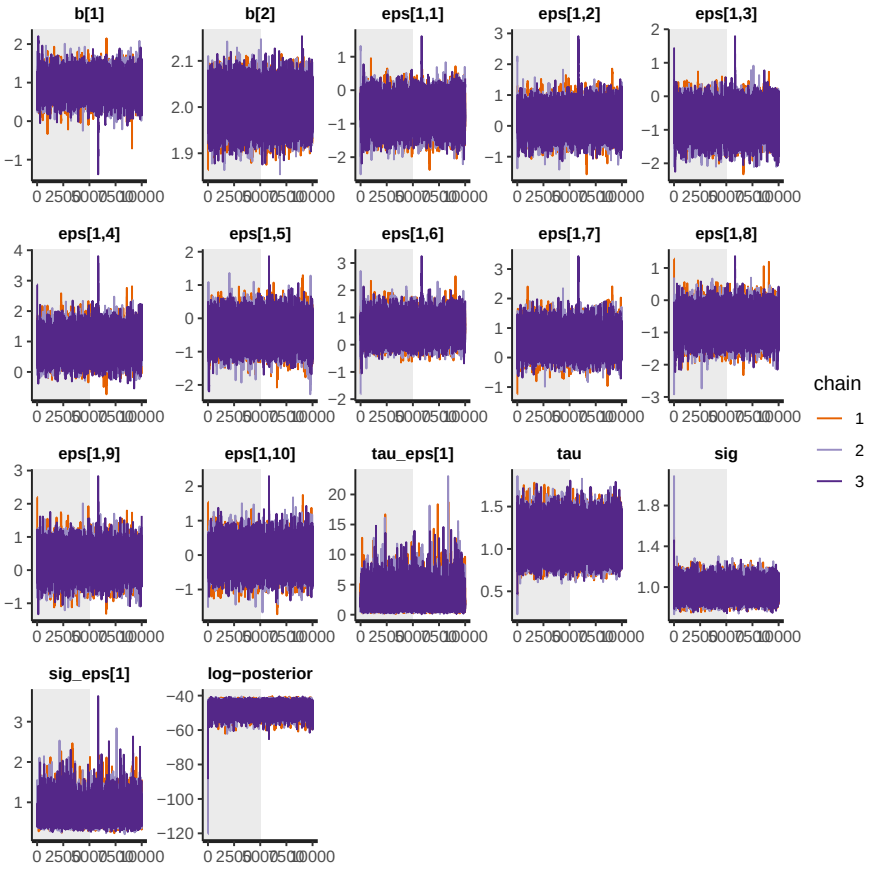
```
> S2.v2.2.stanfit<- sampling(S2.v2.stanmod,
+                            data=S2.v2.2.stan.dat,
+                            init=S2.v2.2.stan.inits,
+                            chains=nchains,
+                            iter=10000,
+                            warmup=5000,
+                            seed=c(20500+5150+24601+2),
+                            refresh=1000,
+                            sample_file="S2.v2.2.chain")
```

Summarize Posterior

```
> summary(S2.v2.2.stanfit)$summary

                  mean      se_mean         sd         2.5%           25%
b[1]         0.93252032 0.0044776300 0.26298868   0.40482372   0.772062411
b[2]         2.00143835 0.0002693998 0.03397691   1.93485179   1.978806475
eps[1,1]    -0.66855037 0.0046621762 0.35765904  -1.39412592  -0.897737010
eps[1,2]     0.15487494 0.0046234582 0.36012420  -0.53899185  -0.080524870
eps[1,3]    -0.70330560 0.0046279153 0.36140698  -1.43103589  -0.932905021
eps[1,4]     0.88674223 0.0050075832 0.37142635   0.17632440   0.639871699
eps[1,5]    -0.37815258 0.0045894244 0.35789248  -1.07907713  -0.610654336
eps[1,6]     0.62210944 0.0048494768 0.35986887  -0.06810327   0.384618069
eps[1,7]     0.61324331 0.0049880424 0.36689838  -0.08061610   0.368928310
eps[1,8]    -0.70583178 0.0046457154 0.36671410  -1.43868525  -0.945345951
eps[1,9]     0.24219848 0.0047774074 0.35351542  -0.42253826   0.009560107
eps[1,10]   -0.01032636 0.0047129755 0.35924645  -0.71044406  -0.247412668
tau_eps[1]   2.47476375 0.0171094364 1.53819678   0.60980366   1.440798539
tau          1.10673897 0.0014159709 0.16710914   0.80500887   0.989646550
sig          0.95880932 0.0006326890 0.07354614   0.82891177   0.906861482
sig_eps[1]   0.72392741 0.0027656474 0.23160754   0.39955620   0.565777622
lp__       -46.88053136 0.0376002311 2.85571637 -53.41443406 -48.573780676
                  50%         75%        97.5%     n_eff      Rhat
b[1]         0.93715382   1.0985897   1.44900090  3449.671 1.0006996
b[2]         2.00163509   2.0241595   2.06820572 15906.441 0.9999639
eps[1,1]    -0.66383157  -0.4320730   0.02304822  5885.198 1.0004204
eps[1,2]     0.14913823   0.3832935   0.87458193  6066.955 1.0002190
eps[1,3]    -0.69805510  -0.4618831  -0.01680559  6098.490 1.0001782
eps[1,4]     0.87579930   1.1182352   1.65380113  5501.601 1.0003694
eps[1,5]    -0.37895836  -0.1445860   0.32357306  6081.192 1.0000844
eps[1,6]     0.61427744   0.8522286   1.33989170  5506.794 1.0006018
eps[1,7]     0.60194231   0.8505786   1.36140340  5410.424 1.0002857
eps[1,8]    -0.70011905  -0.4600807  -0.01805595  6230.889 1.0001800
eps[1,9]     0.23086703   0.4687510   0.95169019  5475.606 1.0004312
eps[1,10]   -0.01209436   0.2195129   0.70392322  5810.247 1.0002599
tau_eps[1]   2.14105165   3.1239816   6.26389198  8082.624 0.9999438
tau          1.09953349   1.2159566   1.45540341 13928.096 1.0000068
sig          0.95366484   1.0052173   1.11455029 13512.606 0.9999836
sig_eps[1]   0.68341802   0.8331024   1.28057494  7013.128 0.9999822
lp__       -46.47033349 -44.8303755 -42.42237312  5768.317 1.0000823
```

```
> traceplot(S2.v2.2.stanfit,
+           pars=names(S2.v2.2.stanfit),
+           inc_warmup=TRUE)
```



S6.8 S2 Scenario 3 sig.eps = 10sig

S6.9 Prepare Data List to Pass to Stan

```
> S2.v2.3.stan.dat<- S2.v2.1.stan.dat
> S2.v2.3.stan.dat$y<- s2y2.gD$y3 ## sig.eps = 10 * sig
```

Prepare Initial Values List to Pass to Stan

Use the `nlme` package to fit the model using restricted/residual maximum likelihood (REML). Results are used to generate initial values for Stan.

```
> s2y3.mod<- nlme::lme(y3 ~ x,
+                      random = ~1 | unit,
+                      data=s2y3.gD)
```

```
> set.seed(20500 + 5150 + 24601 + 3)
>
> ## b[p]
> b<- mvtnorm::rmvnorm(n=nchains,
+                      nlme::fixef(s2y3.mod),
+                      3*vcov(s2y3.mod))
>
> ## eps[q,m]
> mu_eps<- t(nlme::ranef(s2y3.mod))
> eps<- list(matrix(NA,q,m),matrix(NA,q,m),matrix(NA,q,m))
> Sig_eps<- unclass(nlme::getVarCov(s2y3.mod, type="random.effects"))
> for(c in 1:nchains) {
+     for(i in 1:m) {
+         eps[[c]][,i]<- mvtnorm::rmvnorm(n=q,mu_eps[,i],3*Sig_eps)
+     }
+ }
>
> ## tau_eps[q] = 1/sig.eps^2
> tau_eps<- 1/100
>
> ## tau
> tau<- 1.0
>
> S2.v2.3.stan.inits<- list(
+     list(b=b[1,],
+          eps=eps[[1]],
+          tau_eps=as.array(tau_eps,1),
+          tau=tau),
+     list(b=b[2,],
+          eps=eps[[2]],
+          tau_eps=as.array(tau_eps,1),
+          tau=tau),
+     list(b=b[3,],
+          eps=eps[[3]],
+          tau_eps=as.array(tau_eps,1),
+          tau=tau))
```

Run (R) Stan

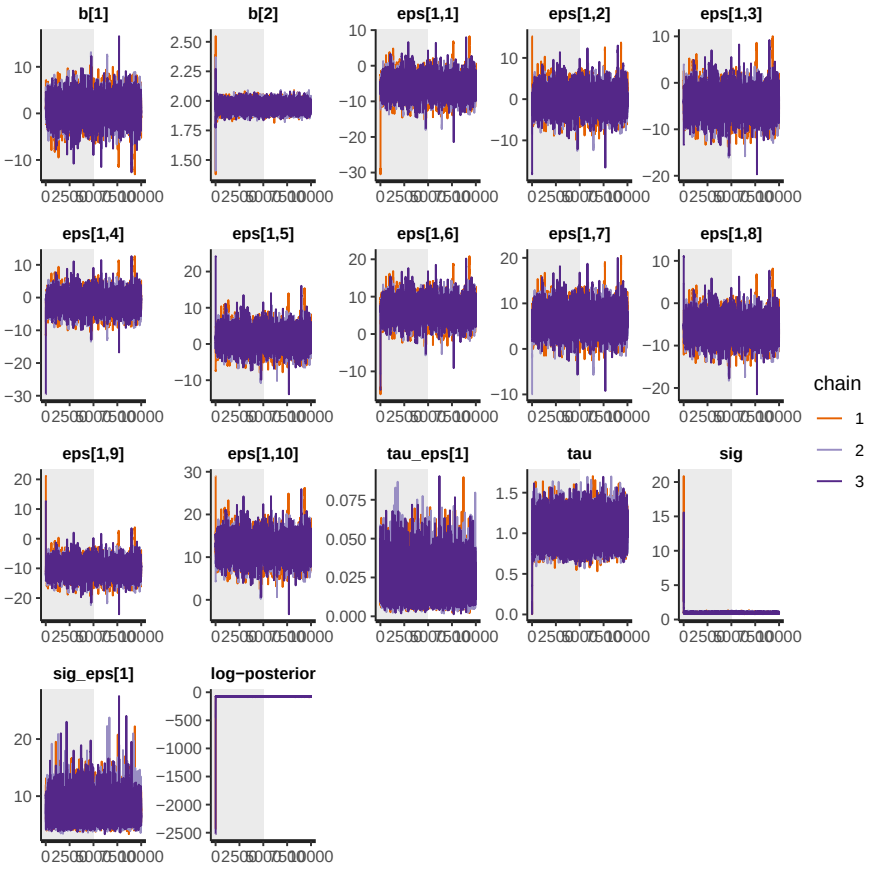
```
> S2.v2.3.stanfit<- sampling(S2.v2.stanmod,
+                            data=S2.v2.3.stan.dat,
+                            init=S2.v2.3.stan.inits,
+                            chains=nchains,
+                            iter=10000,
+                            warmup=5000,
+                            seed=c(20500+5150+24601+3),
+                            refresh=1000,
+                            sample_file="S2.v2.3.chain")
```

Summarize Posterior

```
> summary(S2.v2.3.stanfit)$summary

                  mean      se_mean         sd          2.5%          25%
b[1]         0.90956904 0.0628866512 2.42911434  -3.686877513  -0.60185047
b[2]         1.95581534 0.0004561350 0.03539231   1.885249500   1.93249925
eps[1,1]    -5.69635976 0.0626427364 2.44260696 -10.604352631  -7.21591943
eps[1,2]    -0.10231517 0.0628973552 2.44504769  -4.980396663  -1.63890679
eps[1,3]    -4.25022482 0.0627482057 2.44236874  -9.159743128  -5.77070172
eps[1,4]    -1.18691182 0.0629999714 2.44371181  -6.089384385  -2.71314994
eps[1,5]     1.63851597 0.0628946642 2.44458713  -3.294910021   0.12257386
eps[1,6]     6.44191044 0.0630416553 2.44556934   1.507512947   4.93056327
eps[1,7]     6.47826738 0.0627936919 2.44677843   1.545367178   4.95505180
eps[1,8]    -5.72460060 0.0627743609 2.44431614 -10.617167200  -7.24344504
eps[1,9]   -10.02760066 0.0628676164 2.44375565 -14.964268575 -11.55225587
eps[1,10]   12.28755184 0.0629854387 2.44851881   7.384351926  10.76914467
tau_eps[1]   0.02184313 0.0001334268 0.01006646   0.006994025   0.01444506
tau          1.03040005 0.0020261367 0.15350178   0.753166677   0.92331796
sig          0.99336897 0.0009952197 0.07442203   0.859109277   0.94057838
sig_eps[1]   7.36902625 0.0299662144 1.93498518   4.686703203   6.02593098
lp__       -74.39166388 0.0461781158 2.76276385 -80.762333073 -76.00246754
                  50%          75%        97.5%    n_eff      Rhat
b[1]         0.89008837   2.39788167   5.80737314 1492.035 1.0002959
b[2]         1.95587679   1.97910484   2.02447756 6020.478 1.0001368
eps[1,1]    -5.68884594  -4.15221503  -1.08229095 1520.428 1.0002461
eps[1,2]    -0.07988634   1.42772209   4.53344768 1511.158 1.0003162
eps[1,3]    -4.23580029  -2.71960583   0.36277402 1515.025 1.0002796
eps[1,4]    -1.16735799   0.34388176   3.45829609 1504.594 1.0002792
eps[1,5]     1.64469495   3.19414029   6.27112781 1510.718 1.0002947
eps[1,6]     6.45173741   7.98271509  11.09262211 1504.890 1.0002470
eps[1,7]     6.50240865   8.00131860  11.11062210 1518.299 1.0002771
eps[1,8]    -5.70827123  -4.18079706  -1.07935820 1516.178 1.0002958
eps[1,9]   -10.01050759  -8.48202259  -5.32817475 1510.990 1.0003265
eps[1,10]   12.30147858  13.82724483  16.94367193 1511.216 1.0002958
tau_eps[1]   0.02028210   0.02753922   0.04552659 5692.027 1.0006400
tau          1.02011847   1.13034238   1.35488734 5739.702 0.9999138
sig          0.99009004   1.04069713   1.15227052 5591.974 0.9999262
sig_eps[1]   7.02172002   8.32032575  11.95739111 4169.572 1.0013608
lp__       -74.03747696 -72.38375960 -70.05221063 3579.440 1.0009375
```

```
> traceplot(S2.v2.3.stanfit,
+           pars=names(S2.v2.3.stanfit),
+           inc_warmup=TRUE)
```



S6.10 Saving Coda Objects to File

Optional.

```
> S2.v2.1.stanfit.coda<- As.mcmc.list(S2.v2.1.stanfit)
> S2.v2.2.stanfit.coda<- As.mcmc.list(S2.v2.2.stanfit)
> S2.v2.3.stanfit.coda<- As.mcmc.list(S2.v2.3.stanfit)
> save(list=c("S2.v2.1.stanfit.coda",
+             "S2.v2.2.stanfit.coda",
+             "S2.v2.3.stanfit.coda"),
+      file="S2.v2.stanfit.coda.RData")
```

```
> ## clean up
> detach(package:rstan)
> detach(package:StanHeaders)
> detach(package:ggplot2)
```

```
> rm(list=ls()) ## careful
```