When Models Get Too Large:
Estimability in the Gompertz Stationary and State Space Density Dependence Models with a Covariate

A Thesis

Presented in Partial Fulfilment of the Requirements for the

Degree of Master Science

with a

Major in Statistics

in the

College of Graduate Studies

University of Idaho

by

Cara E. Leatherman

Major Professor: Brian Dennis, Ph.D.

Committee Members: Michelle Wiest, Ph.D.; Ryan Long, Ph.D.

Department Administrator: Christopher Williams, Ph. D.

May 2016

Authorization to Submit Thesis

This thesis of Cara E. Leatherman, submitted for the degree of Master Science with a major in Statistics and titled "When Models Get Too Large: Estimability in the Gompertz Stationary and State Space Density Dependence Models with a Covariate," has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor: _____Date:_____
Brian Dennis, Ph.D.

Committee Members: _____Date:_____
Michelle Wiest, Ph.D.

_____Date:_____
Ryan Long, Ph.D.

Department Administrator: _____Date:_____
Christopher Williams, Ph.D.

Abstract

We studied the limits of estimability of stochastic versions of the Gompertz model of density dependent population growth when the models are expanded to include an environmental covariate. The stochastic versions were the Gompertz model with process noise (GPN) and the Gompertz state space model (GSS) containing both process noise and observation error. Simulation trials and maximum likelihood estimates of the parameter values show that when sample size is low (n=10) the addition of the covariate in the GPN model causes estimability to break down, but the GPN model performs adequately for longer time series. In most cases studied, the GSS model with a covariate has extremely high estimate variance, estimates often covering the entire range of possible values of the parameter of interest. These results represent severe limitations to the use of covariates with the GPN and GSS models and do not bode well for larger state space and other hierarchical models used in modern statistics.

Acknowledgements

First and foremost, I would like to thank my major advisor, Dr. Brian Dennis. His support and assistance in creating this work has been invaluable from the moment I decided to write a thesis through the final revision. I would also like to thank my committee, Drs. Michelle Wiest and Ryan Long. Without them I would never have recognized the importance of my little simulation study. Thank you.

Table of Contents

List of Figures

CHAPTER 1

Introduction

Density dependence has been extremely important in ecology since its proposal as a population growth mechanism (for reviews, see Bellows 1981, Fowler 1987, Bjørnstad and Grenfell 2001). Density dependence models have been used as management aids for integral tasks such as predicting chances of population extinction (for example, Krkošek et al. 2007). Model complexity has slowly increased over the decades. Adding stochasticity due to environmental effects or individual variation led to a multitude of growth models (for example, Dennis and Taper 1994, Ives et al. 2003, Turchin 2003) that have been used extensively. To more accurately represent real-world population growth and surveying methodology, observation error has also been added to the models. Such models that include observation error as well as process noise are known as state space models (de Valpine 2002, de Valpine and Hastings 2002, Clark and Bjørnsted 2004, Dennis et al. 2006).

The addition of covariates to models has also been very instrumental in ecological modeling, especially in support of management decisions. Covariates can include climate or habitat changes, other species, or any other important effect to the population of interest. They are commonly used in model selection proceedures to determine what environmental effects are most integral to population dynamics and survival. Covariates have proved useful for model selection in studies as diverse as elk calf mortality (Singer et al. 1997), spotted owl recovery (Franklin et al. 2000), kit fox survival chances (Dennis and Otten 2000), mule deer population trends (Peek et al. 2002), and probabilities of survival in various salmonid species (McClure et al. 2003).

Complex models come at a cost, however. While neglecting model observation error has been shown to have its own price in biased estimates (Shenk 1998, Holmes 2001, De Valpine and Hastings 2002, Freckleton 2006) and increased Type 1 errors in hypothesis testing (Freckleton 2006), state space models appear to be approaching the limits of estimability (Dennis et al. 2006, Knape 2008, White 2012). Adding covariates to a model already at the

edge of estimability has not been researched and may be suspect.

Here, we add a covariate to the Gompertz process noise (GPN) and Gompertz state space (GSS) models developed by Dennis et al. (2006), creating the Gompertz process noise with covariate (GPNc) and Gompertz state space with covariate (GSSc) models. Using simulation and maximum likelihood estimation on the GPNc and GSSc, we attempt to find whether estimation for the models breaks down. The simulation results presented here reveal constraints on estimability of model parameters that should be taken into account when the models are used for describing ecological data.

CHAPTER 2

Methods

2.1 Model form and notation

The models used in this report are founded on the Gompertz density dependence model. The Gompertz model has often been used for population abundance modeling. We used both the GPN and GSS models, as defined in Dennis et al (2006). The GPN is a discrete time model with the following form:

$$N_t = N_{t-1} \exp\left(a + b \ln N_{t-1} + E_t\right) \tag{2.1}$$

Here, N_t is the population size at time t. The constants a and b are defined below. The error term, or process noise, at time t is E_t and is distributed normally with a mean of 0 and variance σ^2 (written $E_t \sim \mathcal{N}(0, \sigma^2)$). To make calculations easier, we take the natural log of the equation to find:

$$X_t = X_{t-1} + a + bX_{t-1} + E_t = a + cX_{t-1} + E_t \tag{2.2}$$

where $X_t = \ln N_t$ and $c = b + 1$. The stationary mean and variance of X_t are functions of a, c, and σ^2:

$$\mathbb{E}(X_\infty) = \frac{a}{1-c} \tag{2.3}$$

$$\mathbb{V}(X_\infty) = \frac{\sigma^2}{1-c^2} \tag{2.4}$$

For all models used, $-1 < c < 1$ and $a > 0$. The parameter c can be viewed as an inverse mesure of the strength of density dependence in the model, or the speed with which logarithmic population size X_t reaches its stationary distribution. As c approaches 1, the stationary mean becomes undefined and we consider the population to be density independent (exponential growth).

The GSS model, as defined by Dennis et al. (2006), adds an observation error term to the GPS to account for imperfect sampling. Let Y_t be the log observed or estimated population size at time t. The GSS model defines Y_t by:

$$Y_t = X_t + F_t \tag{2.5}$$

where the observation error at time t, $F_t \sim \mathcal{N}(0, \tau^2)$. On the original scale of population size N_t, F_t follows a lognormal distribution. A lognormal observation error distribution is ecologically realistic (Dennis et al. 2006).

We expand the GPN and the GSS models to include an environmental covariate in the following way. We redefine a using the covariate term, under the assumption that an environmental change would likely change the location of the stationary mean. Thus, we substitute $\exp(\beta_0 + \sum_{i=1}^{p} \beta_i W_{i,t})$ for a, where any number of covariates can be added with any time lag $(t, t-1, \ldots)$ deemed ecologically important. Here, β_0 takes the basic definition of a previously, while β_i describes the influence of the i_{th} covariate, $W_{i,t}$. Thus,

$$X_t = \exp(\beta_0 + \sum_{i=1}^{p} \beta_i W_{i,t}) + cX_{t-1} + E_t \tag{2.6}$$

for the GPNc, and

$$Y_t = X_t + F_t \tag{2.7}$$

for the GSSc, as previously.

2.2 Likelihood function

The time series data are population abundances. These are log-transformed to find X_t or Y_t, depending on whether the abundances are censused or estimated. Data on covariates are added to the model as $W_{i,t}$ quantities. We assume that the data arise from either model as described by equations 2.6 and 2.7 and are sampled from a population undergoing density

dependent growth.

In order to find estimates for the parameters $\beta_0, \beta_1, c, \sigma^2, \tau^2$, likelihood functions are necessary. The likelihood functions take different forms for the GPNc and the GSSc models.

The GPN is an autoregressive time series model of order 1 [AR(1) process], meaning that each observation is dependent only on the previous time period's observation. As the stochasticity of the model is found in the normally distributed process noise, the autoregressive function for X_t itself comes from the normal distribution. It has an expected value of $a - cX_{t-1}$ and variance σ^2. In the GPNc, a is replaced by $\exp(\beta_0 + \sum_{i=1}^{p} \beta_i W_{i,t})$ so

$$X_t \mid X_{t-1} \sim \mathcal{N}\left(\exp(\beta_0 + \sum_{i=1}^{p} \beta_i W_{i,t}) - cX_{t-1}, \sigma^2\right) \tag{2.8}$$

As shown by Dennis et al. (2006), the GSS model does not have a simple autoregressive form. Y_t comes from a normal distribution, but due to the combination of process and observation noise, depends on $Y_{t-1}, Y_{t-2}, \ldots, Y_0$.

$$Y_t \mid Y_{t-1}, Y_{t-2}, \ldots, Y_0 \sim \mathcal{N}(m_t, v_t^2) \tag{2.9}$$

The mean and variance are found recursively by Kalman iteration (Dennis et al. 2006). For the GSSc model, the Kalman iterations take the form,

$$m_t = \exp(\beta_0 + \sum_{i=1}^{p} \beta_i W_{i,t}) + c\left[m_{t-1} + \frac{v_{t-1}^2 - \tau^2}{v_{t-1}^2}(y_{t-1} - m_{t-1})\right] \tag{2.10}$$

$$v_{t-1}^2 = c^2 \frac{v_{t-1}^2 - \tau^2}{v_{t-1}^2} + \sigma^2 + \tau^2 \tag{2.11}$$

When the population counts come from the stationary distribution, as was the case in this study, the recursion is started at the stationary mean and variance: $m_0 = \exp(\beta_0 + \beta_i W_{i,t})/(1-c)$ and $v_0^2 = [\sigma^2/(1-c^2)] + \tau^2$. For other cases and derivations, see Dennis et al. (2006).

The one-step probability distribution functions for the GPNc and GSSc models respec-

tively become

$$f(x_t \mid x_{t-1}) = (\sigma^2 2\pi)^{-1/2} \exp\left\{ -\frac{\left[x_{t-1} - \exp\left(\beta_0 + \sum_{i=1}^{p} \beta_i W_{i,t}\right) - c x_{t-1}\right]^2}{2\sigma^2} \right\} \tag{2.12}$$

$$f(y_t \mid y_{t-1}, y_{t-2}, \ldots, y_0) = (v_t^2 2\pi)^{-1/2} \exp\left[-\frac{(y_t - m_t)^2}{2v_t^2} \right]. \tag{2.13}$$

Y_0 is also normally distributed with pdf

$$f(y_0) = (v_0^2 2\pi)^{-1/2} \exp\left[-\frac{(y_0 - m_0)^2}{2v_0^2} \right] \tag{2.14}$$

The likelihood functions for the GPNc and GSSc models are the joint pdfs:

$$L(x \mid \beta_0, \beta_i, c, \sigma^2) = \prod_{t=1}^{n} (\sigma^2 2\pi)^{-1/2} \exp\left\{ -\frac{\left[x_{t-1} - \exp\left(\beta_0 + \sum_{i=1}^{p} \beta_i W_{i,t}\right) - c x_{t-1}\right]^2}{2\sigma^2} \right\} \tag{2.15}$$

$$L(y \mid \beta_0, \beta_i, c, \sigma^2, \tau^2) = (v_0^2 2\pi)^{-1/2} \exp\left[-\frac{(y_0 - m_0)^2}{2v_0^2} \right] \prod_{t=1}^{n-1} (v_t^2 2\pi)^{-1/2} \exp\left[-\frac{(y_t - m_t)^2}{2v_t^2} \right] \tag{2.16}$$

To perform maximum likelihood estimation, the log likelihood is used.

2.3 Simulations

We conducted simulations of time series data using both the GPNc and GSSc using a single covariate to evaluate estimability of the models in the simplest case. Each set contained 1000 simulated time series. Maximum likelihood estimates for all parameters were obtained for each simulated time series and results were plotted. The plots were used to analyze estimability for each situation simulated.

R version 3.2.2 was used for all simulations and optimizations. We used the Nelder-Mead algorithm in the optim() function for maximum likelihood estimation. A seed of 100 was used for each set of 1000 simulations. The single covariate for each model was randomly

generated from a standard normal distribution. Values for the β_0 and β_1 parameters were found using the equation for the stationary mean given the simulated starting population size and strength of covariate influence. Environmental error (GPNc, GSSc) and observation error (GSSc only except in specific cases) were included in each simulated time series as per Equations 2.2 and 2.5. Code used for β_0 and β_1 solutions, simulation, and maximum likelihood estimation can be found in Appendix A.

Maximum likelihood estimates of all parameters in the GPNc model were found from time series data simulated with only process noise (GPN model). The GPNc model was also used on data simulated with observation error to determine levels of bias and variance inflation when using a covariate in the model. For the GSSc model, data with observation error was also used to find maximum likelihood estimates of all parameters.

Default values for the simulations are as follows. Data are simulated with a starting population size of 1000. Covariate influence of 1 standard deviation change in covariate resulting in a 20% increase in population size (resulting in a β_0 value of 0.026) is used. The default c value is -0.5. σ^2 and τ^2 values are both 0.5. Each parameter was adjusted individually from these defaults to determine its effect on estimability.

For reasons of space in this paper, c is our main parameter of interest. The parameter c, as the strength of density dependence, is critical to predicting underlying dynamical behavior important to policy and managment. Other parameters are also important. For instance, probabilistic properties of the population, such as the chances of recovery or the risks of quasi-extinction, revolve on the estimability of σ^2. We show such additional results in Appendix B.

CHAPTER 3

Results

3.1 Gompertz process noise with covariate simulations

As the value of c changes through $-.9, -.5, 0, .5$, and $.9$, we observe no substantial difference in the ability of the GPNc to estimate the parameters (Figure 3.1). There is no noticeable difference in variance associated with a changing value of c. However, when $c = .9$, there is some bias as the mean simulation estimated value is 0.737. There is also some bias when $c = -.9$, showing a mean value of -0.828. The bias might be due to being near the edge of supported c values, artificially creating a situation where the means are unlikely to be correct due to the truncation of simulated c estimates at -1 and 1.

Using the values of c shown above but changing the sample size (time series length) to 10 results in more variance and bias in all cases of c (Figure 3.2). In many cases there are estimates ranging over the complete space of -1.0 to 1.0, indicating questionable estimability. As sample size becomes small the model is less capable of accurately estimating the parameters.

Changing the influence strength of the covariate shows that as strength of influence increases, whether positive or negative, there is less variance. We simulated a 50% increase/decrease in population carrying capacity for a single standard deviation change in covariate strength as well as a 20% increase/decrease. Both 50% cases appear to have about the same variance. The 20% cases also appear very similar but have a very slightly higher variance than the 50% cases. In all cases, however, the simulations converge about the true value of $c = -.5$, indicating no estimability or bias issues in the GPNc model due to changing influence of the covariate.

Decreasing the size of the population does not appear to affect the variance or accuracy of the simulated estimates. Using population sizes of 100 and 1000, we find that in both cases, the estimates converge on the true value of $c = -.5$, indicating that the model is

equally useful for either population size.

3.2 Gompertz process noise with covariate including observation error simulations

Data were simulated with observation error and parameter estimates were found using the GPNc model, though it does not include a term for observation error. We expect increased bias as the unmodeled observation error increases. We see in Figure 3.3, where the unmodeled observation error has increased from 0.05 to 0.5 in 0.05 increments, that bias does increase. There is also a slight increase in variance of the estimates with increasing observation error, but the increased variance is observed mostly in the tails of the estimate distributions.

Because management decisions often involve more than an estimate of the strength of density dependence, we include graphs of how changing observation error affects other parameters and at different values of c in Appendix B.

3.3 Gompertz state space with covariate simulations

Estimation results for the GSSc with a single covariate are much more problematic than seen with the GPNc. We begin to see the problems when the values of c are increased with a time series of length 30 (Figure 3.4). In every situation except $c = -.5$, there are estimates that span the entire support set of c, showing severe lack of estimability. When c is positive, there is almost no precision in the estimates. In fact, when $c = .5$, the estimates have a mean of -0.354 and median of -0.508, almost exactly the opposite of the true value of c. Histograms of the changing values of c have been included in addition to the usual box plots in order to show the patterns of estimate results (Figure 3.5). As indicated by the histograms, the optimization routine appears to have become caught in the local maxima at extreme values of c shown by Dennis et al (2006).

As the length of the time series decreases we find that, unlike with the GPNc, estimability

problems do not overall become more pronounced (Figure 3.6). In all cases, estimates cover the most of the range of c. Even in the best case, when $c = -.9$, 95% of observations cover the interval $(-0.996, -0.102)$, nearly half of all possible values. There is an increasingly high amount of bias as c increases. When c is non-negative, the estimates appear to nearly reverse in signage, as seen when $n = 30$.

Even when the length of the time series is increased to 100, estimability problems are unresolved (Figure 3.7). Positive values of c continue to show huge bias, other than in the $c = 0.9$ case. That $c = 0.9$ shows less bias is likely due to the extreme value effect shown in Figure 3.5. When c is close to 1, the optimization routine becomes more often caught in these local maxima, bringing more estimates closer to 0.9. Variance is very large throughout and worse when c is positive.

The effect of changing the influence of the covariate from an increase/decrease of 20% to an increase/decrease of 50% is, similar to the GPNc model, very minimal in terms of the accuracy of the resulting estimates. While the increase/decrease of 50% appears have slightly less variance, it is perhaps more noteworthy that it seems to also have slightly less bias.

Estimates show very similar variance in population sizes of 100 and 1000. The estimate distributions seem to have longer tails in the smaller population size but slightly more bias in the larger population size.

Because observation error is modeled in the GSSc, we do not expect any difference in bias or variance as observation error is increased. However, Figure 3.8 shows that as observation error increases from 0.05 to 0.5 in .05 increments the bias in c estimates actually decreases. There is pronounced bias in the case of low observation error. Variability is high through all levels of τ^2 and estimates are unlikely to be very trustworthy. Variance appears to increase with increased observation error, but the appearance may be due more to the truncation of c at -1.0 and bias changes than any true increase in variance.

As with the GPNc results, we have included graphs of how changing observation error

affects other parameters and at different values of c in Appendix B.

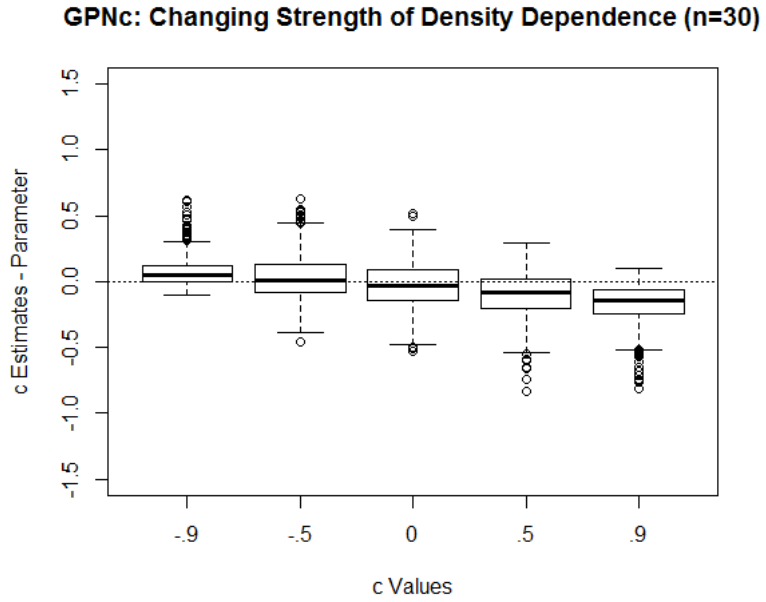


Figure 3.1: Using the GPNc, as c increases from $-.9$ to .9, there is no appreciable difference in the variance of the simulated estimates of c, though some bias is seen in the $-.9$ and .9 results.

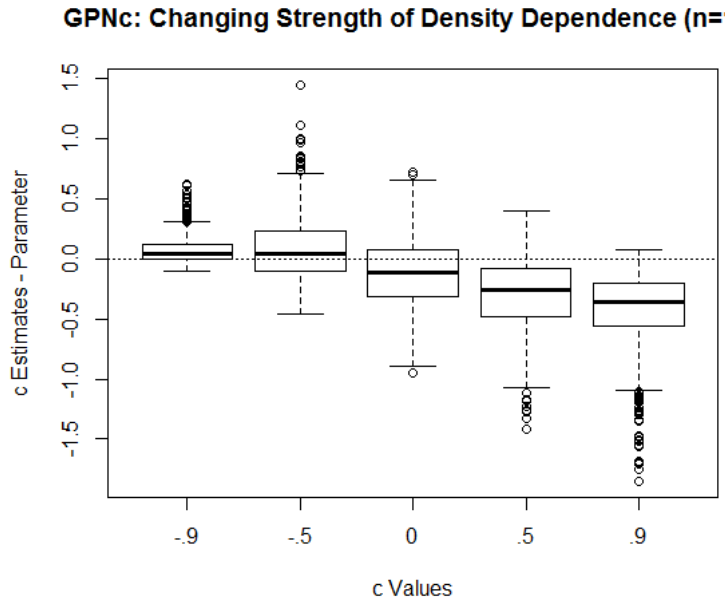


Figure 3.2: When the time series length drops to 10 using the GPNc, there are more estimability issues with the GPNc. There is increased bias and variance as well as cases where c estimates cover a large portion of the supported values of c.

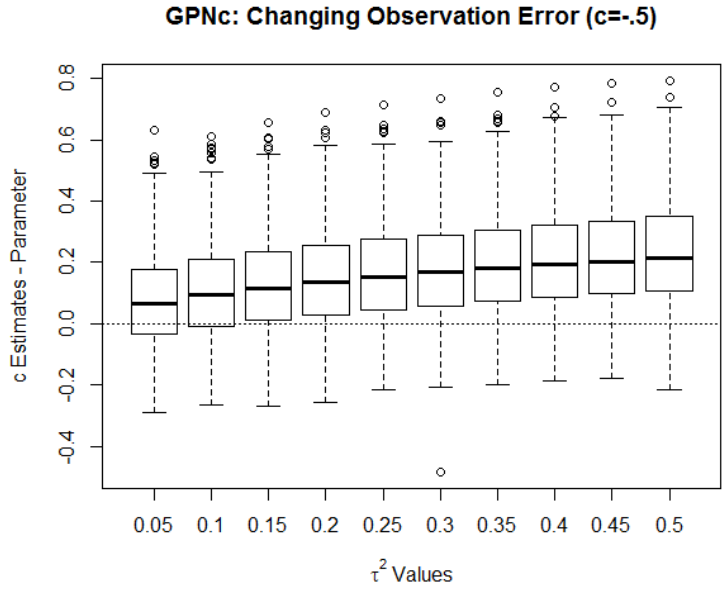


Figure 3.3: When unmodeled observation error is added to the GPNc with $c = -0.5$ in 0.05 increments from 0.05 to 0.5, we see increasing bias in the c estimates. Variance remains similar throughout.

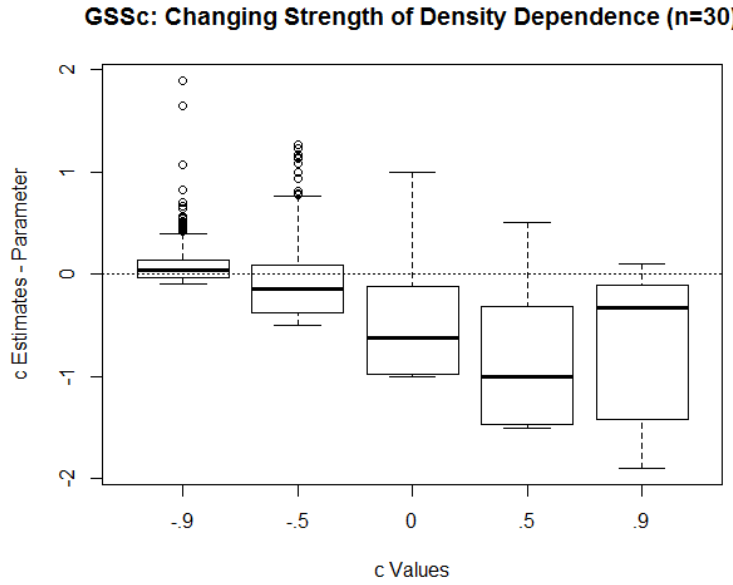


Figure 3.4: Changing values of c in the GSSc model show estimability problems. The model does not appear to be capable of estimating all parameters adequately, except perhaps in the cases of $c = -.9$ and $c = -.5$ (a and b).

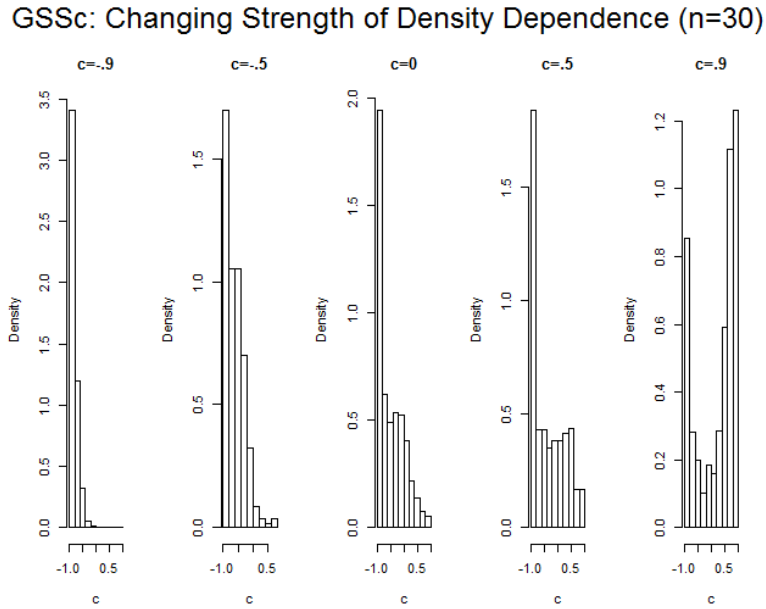


Figure 3.5: Changing values of c in the GSSc model show estimability problems. The model does not appear to be capable of estimating all parameters adequately, except perhaps in the cases of $c = -.9$ and $c = -.5$ (a and b). Histograms show the pattern of estimation problems.

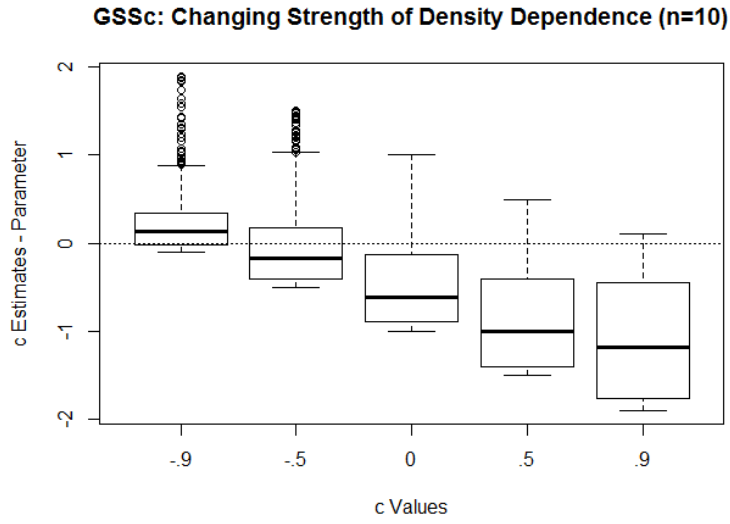


Figure 3.6: When the length of the time series using the GSSc model is reduced to 10, estimability problems are clear, growing more pronounced as c increases. When c is non-negative, estimates cover most of the range of c and bias becomes pronounced enough that any estimate achieved is all but meaningless.

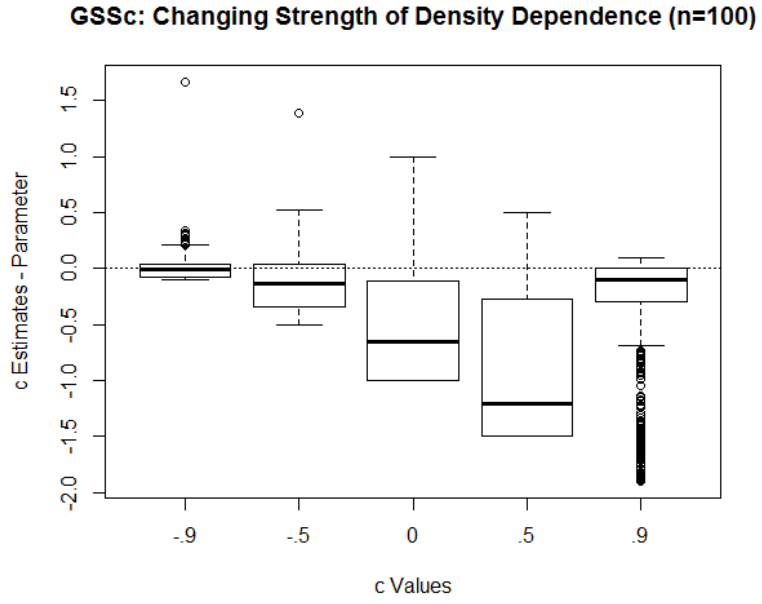


Figure 3.7: When the length of the time series using the GSSc model is increased to 100, estimability problems are still clear, growing overall more pronounced as c increases. When c is non-negative, estimates cover most of the range of c and bias becomes pronounced enough that any estimate achieved is all but meaningless.

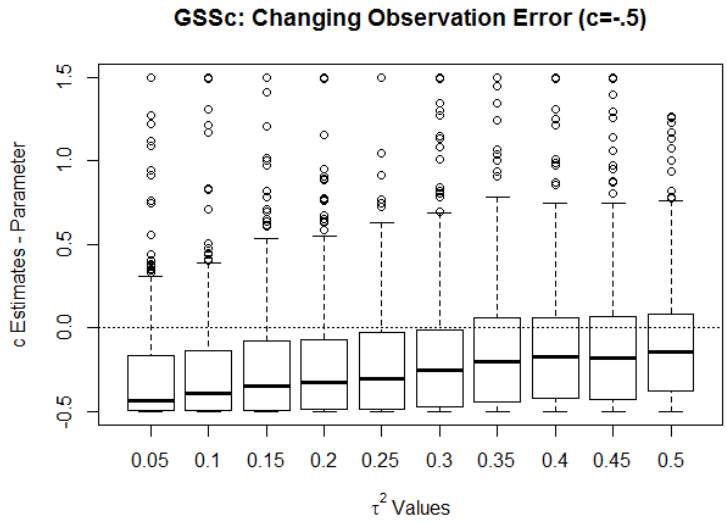


Figure 3.8: As observation error is added to the GSSc when $c = -.5$ in 0.05 increments from 0.05 to 0.5, there are extreme estimates of c throughout, along with decreasing bias and increasing variance as τ^2 increases.

CHAPTER 4

Discussion

The simulation results show that adding a covariate to the stochastic versions of the Gompertz model can lead to problems in estimability. It is worth noting that this covariate adds only one parameter to the estimability study conducted by White et al. (2012). In the case of the GPNc, estimability problems can be solved with a larger sample size, though achieving a sample size much greater than 10 can be difficult in many ecological applications. For the GSSc, where the estimability problems are particularly severe, there is no clear solution. Presumably, the addition of the covariate has created a model with too many parameters. As sample sizes up to 100 continued to exhibit estimability problems, we conclude that the problems are likely model based rather than due to a lack of data.

Potential methods of dealing with the estimability problems presented in this paper were not specifically addressed in this study. One method includes dropping parameters when they are not of direct interest. For example, zeroing out the intercept parameter $beta_0$ simplifies the model, possibly making it more viable to add a covariate. Similarly, if one is not directly concerned with whether the population is undergoing density dependent growth, one might chose to use a model that does not include the density dependence term (Holmes 2007). Replicated sampling has shown to have increased power in some circumstances (Dennis et al. 2010, See and Holmes, 2015) and may be a useful option to improve estimates. Alternatively, Hostetler and Chandler (2015) have proposed a different set of models and tests that appear to have improved estimability.

Models with density dependence should only be used with populations that appear to exhibit fairly strong density dependence. We chose to use $c = -.5$ as our default parameter for simulations because it is the value of c that behaves best in most circumstances. In situations where the population is starting much lower than the carrying capacity and going through clear logistic growth, the model may behave better, but that case is not investigated here. When a population starts out at or around carrying capacity, we recommend that

the user check that there is a clear return after each departure from carrying capacity to determine whether density dependent growth is the likely growth mechanism.

Using more representative starting values for the optimization routine can help with convergence, although such starting values will not address estimability. For all maximum likelihood estimates in this study, the same starting values were used for the optimization routine. In modeling real populations, a researcher should use starting values as close as possible to the real parameter values. Using values close to the real ones helps prevent the routine from getting caught in local maxima that may not be the global estimate. When one can estimate starting values that are more likely to be similar to parameter values using the sampled time series, one should do so.

Because the GSSc performs so poorly, a researcher might choose to instead use the GPNc and leave observation error unmodeled. In cases where the observation error is small, using the GPNc may be a valid option (Figure 3) as bias is relatively small in these cases. However, we see that bias increased for all parameters as unmodeled observation error increases (Appendix B). In most cases, so do estimate distribution variances. The most dramatic case of increased bias is the estimate for σ^2. When one is, for example, modeling the probability of population extinction, a highly biased σ^2 may be unacceptable. Finding confidence intervals, whether through bootstrapping or MCMC using data cloning (as described by Ponciano et al. 2009), should assist the researcher in determining how precise the results might be.

Additional covariates are likely to compound the problems shown here. Modern models can include dozens, if not hundreds, of covariates and should be used with extreme caution. Any maximum likelihood optimization routine is likely to give the user an estimate for each parameter, but it is difficult to know if the estimates thus obtained represent the real population dynamics. Bayesian statistics, with its flexibility and prevalence in heirarchical modeling, may seem to be a solution. However, analyzing using Bayesian techniques when there are estimability problems will likely result in getting back the priors rather than any meaningful posterior parameter distributions. Neither additional covariates nor Bayesian

techniques were directly investigated here, however.

When creating a new model or investigating an existing model, a researcher should be aware of "red flags" that indicate possible estimability problems. Confidence intervals that are wider than expected may indicate such a problem. Profile likelihood plots that are either ridge-shaped or jagged and/or multimodal should prompt further investigation. While some packages exist to create profile plots for simple and common models in R, for complex models the researcher will likely have to code them. Appendix C.1 contains an example of profile plot code fitting the GSSc model to real data from the Peek et al. (2002) mule deer study. This study is also used to show an example of profile plots show potential estimability problems. Were a researcher to get similar plots, further examination of the model is recommended. Estimability can be evaluated using a simulation study similar to the one presented here or using data cloning, as described by Lele et al. (2010).

This report indicates that caution should be used with large heirarchical models in general. We only examined one model, with two variants, among the many in use. Modern computing power has allowed incredibly complex models, held back only by the time a researcher is willing to allot to the computation. This increased computation seems a blessing, but our simulations show that the ability to compute estimates for highly complex models should be approached very carefully. Other large state space and hierarchical models might be equally at risk for estimability problems. Certainly, more research is needed to determine the points at which other models break down. We see nothing intrinsic to Gompertz models that makes them susceptible to estimability problems and expect such problems to pervade more complex models of other forms.

References

[1] Bellows, T. S. 1981. The descriptive properties of some models for density dependence. Journal of Animal Ecology 50:139-156.

[2] Bjornstad, O. N., and B. T. Grenfell. 2001. Noisy clockwork: Time series analysis of population fluctuations in animals. Science 293:638-643.

[3] Clark, J. S., and O. N. Bjornstad. 2004. Population time series: Process variability, observation errors, missing values, lags, and hidden states. Ecology 85:3140-3150.

[4] de Valpine, P. 2002. Review of methods for fitting time-series models with process and observation error and likelihood calculations for nonlinear, non-Gaussian state-space models. Bulletin of Marine Science 70:455-471.

[5] de Valpine, P., and A. Hastings. 2002. Fitting population models incorporating process noise and observation error. Ecological Monographs 72:57-76.

[6] Dennis, B., and M. R. M. Otten. 2000. Joint effects of density dependence and rainfall on abundance of San Joaquin kit fox. Journal of Wildlife Management 64:388-400.

[7] Dennis, B., J. M. Ponciano, S. R. Lele, M. L. Taper, and D. F. Staples. 2006. Estimating density dependence, process noise, and observation error. Ecological Monographs 76:323-341.

[8] Dennis, B., J. M. Ponciano, and M. L. Taper. 2010. Replicated sampling increases efficiency in monitoring biological populations. Ecology 91:610-620.

[9] Dennis, B., and M. L. Taper. 1994. Density-dependence in time-series observations of natural populations - estimation and testing. Ecological Monographs 64:205-224.

[10] Fowler, C. 1987. A review of density dependence in populations of large mammals. Pages 401-441 in H. H. Genoways, editor. Current Mammalogy. Springer US.

[11] Franklin, A. B., D. R. Anderson, R. J. Gutierrez, and K. P. Burnham. 2000. Climate, habitat quality, and fitness in Northern Spotted Owl populations in northwestern California. Ecological Monographs 70:539-590.

[12] Freckleton, R. P., A. R. Watkinson, R. E. Green, and W. J. Sutherland. 2006. Census error and the detection of density dependence. Journal of Animal Ecology 75:837-851.

[13] Holmes, E. E. 2001. Estimating risks in declining populations with poor data. Proceedings of the National Academy of Sciences of the United States of America 98:5072-5077.

[14] Holmes, E. E., J. L. Sabo, S. V. Viscido, and W. F. Fagan. 2007. A statistical approach to quasi-extinction forecasting. Ecology Letters 10:1182-1198.

[15] Hostetler, J. A., and R. B. Chandler. 2015. Improved state-space models for inference about spatial and temporal variation in abundance from count data. Ecology 96:1713-1723.

[16] Ives, A. R., B. Dennis, K. L. Cottingham, and S. R. Carpenter. 2003. Estimating community stability and ecological interactions from time-series data. Ecological Monographs 73:301-330.

[17] Knape, J. 2008. Estimability of density dependene in models of time series data. Ecology 89:2994-3000.

[18] Krkošsek, M., J. S. Ford, A. Morton, S. Lele, R. A. Myers, and M. A. Lewis. 2007. Declining wild salmon populations in relation to parasites from farm salmon. Science 318:1772-1775.

[19] Lele, S. R., K. Nadeem, and B. Schmuland. 2010. Estimability and Likelihood Inference for Generalized Linear Mixed Models Using Data Cloning. Journal of the American Statistical Association 105:1617-1625.

[20] McClure, M. M., E. E. Holmes, B. L. Sanderson, and C. E. Jordan. 2003. A large-scale, multispecies status, assessment: Anadromous salmonids in the Columbia River Basin. Ecological Applications 13:964-989.

[21] Peek, J. M., B. Dennis, and T. Hershey. 2002. Predicting population trends of mule deer. Journal of Wildlife Management 66:729-736.

[22] Ponciano, J. M., M. L. Taper, B. Dennis, and S. R. Lele. 2009. Hierarchical models in ecology: confidence intervals, hypothesis testing, and model selection using data cloning. Ecology 90:356-362.

[23] See, K. E., and E. E. Holmes. 2015. Reducing bias and improving precision in species extinction forecasts. Ecological Applications 25:1157-1165.

[24] Shenk, T. M., G. C. White, and K. P. Burnham. 1998. Sampling-variance effects on detecting density dependence from temporal trends in natural populations. Ecological Monographs 68:445-463.

[25] Singer, F. J., A. Harting, K. K. Symonds, and M. B. Coughenour. 1997. Density dependence, compensation, and environmental effects on elk calf mortality in Yellowstone National Park. Journal of Wildlife Management 61:12-25.

[26] Turchin, P. 2003. Complex population dynamics: a theoretical/ empirical synthesis. Princeton University Press, Princeton, New Jersey, USA.

[27] White, K., B. Dennis, P. Joyce, and D. Scarnecchia. 2012. Estimability of parameters in time series population abundance models assessed using data cloning. University of Idaho, Moscow, Idaho, USA.

Appendix A: R Code

R code for finding the solutions for β_0 and β_1, simulations, and estimation are included here.

Appendix A.1. Solving for β_0 and β_1

```
### Solving for values of beta1 and beta0 ###
# To solve for value of beta1 when 1 standard deviation increase
# creates a 25% increase in mean population using the equation for
# the stationary mean with covariate term substituted.
# We are looking at a population with size n and assuming n is the
# mean of the stationary distribution.
# E(n_inf)=a/(1-c)
# a=exp(beta0+beta1*w) where w values are standardized such
# that w_bar=0.
# At the mean of w, a=exp(beta0), so beta0=log(a)
# If we are looking for the effect of a one standard deviation
# change of w that produces some percentage change in the
# mean population size (e.g. from a starting value of 1000,
# we expect a 20% change in population size, so the new
# expected population size is 1200), the mean of w becomes 1,
#  so a_new=exp(beta0+beta1).

# User defined section:
k=1000      # set mean population value
knew= 1200 # set new value of population after effect of covariate
```

```
cc=-0.5       # set value of c of interest

xbar=log(k); aa=xbar*(1-cc); beta0=log(aa)
xnew=log(knew); beta1=log(xnew*(1-cc))-beta0

# print results:
beta0
beta1
```

Appendix A.2. Simulating Datasets

```
### Script to simulate time series data for the GPNc and GSSc ###
# This code allows the user to create any number (km) of
# simulations of any time series length (k) by entering values of
# c, beta0, beta1, sigma_sq, and tau_sq as defined by the user. To
# use data from the GPNc, simply use data created in the
# x_sim_matrix. To use the data from the GSSc, use data created
# in the y_sim_matrix.

## User defined section: ##

k=10                    # User defined # of time series observations
km=1000                 # User defined number of simulations
cc_sim=-0.9             # User defined value of c
beta0_sim=2.574499      # User defined value of beta_0
beta1_sim=0.02605144    # User defined value of beta_1
```

```
sigmasq_sim=.5          # User defined value of sigma_squared
tausq_sim=.5            # User defined value of tau_squared
pop=1000                # User defined starting population size

## Simulation section: ##
# Simulate time series from given covariate/parameter values with
# starting population size of pop.
set.seed(100)
q=k-1                   # Number of time changes
covariate=rnorm(q)      # Simulating covariate from standard normal
x_sim_matrix=matrix(1,nrow=km, ncol=k)
y_sim_matrix=matrix(1,nrow=km, ncol=k)
x_sim_matrix[,1]=log(pop)
y_sim_matrix[,1]=x_sim_matrix[,1]+rnorm(km,0,sqrt(tausq_sim))
for(t in 1:q){
    x_sim_matrix[,t+1]=exp(beta0_sim+beta1_sim*covariate[t])
      +cc_sim*x_sim_matrix[,t]+rnorm(km,0,sqrt(sigmasq_sim))
    y_sim_matrix[,t+1]=x_sim_matrix[,t+1]
      +rnorm(km,0,sqrt(tausq_sim))
}
```

Appendix A.3. Finding MLEs of Simulated Data

```
### Script to find Maximum Likelihood Estimates ###
### of Simulated Data for GPNc and GSSc ###
# This code allows the user to find the maximum likelihood
```

```
# estimates of data simulated using script to simulate time series
# data. It will result in a matrix of MLEs where each row is the
# beta0, beta1, c, sigma_sq, and tau_sq MLE followed by the
# likelihood for the corresponding time series from the
# simulated data.
# To get GPNc results, use the first set of code within the
# estimation section.
# To get GSSc results, use the second set of code within the
# estimtion section.
# To find GPNc with unmodeled observation error, use
# y_sim_matrix data from the script to simulate time series
# data but GPNc code below.

## User defined section: ##
# Set file to save simulation results to if desired:
sink("...")
# Set initial values of parameters for optimization routine (either
# near real values for data where approximate real values may be
# known or at a distance to check for estimability issues)
beta0_0=1           # User defined initial value of beta0
beta1_0=.1          # User defined initial value of beta1
cc_0=.1             # User defined initial value of c
sigmasq_0=1         # User defined initial value of sigma_squared
tausq_0=1           # User defined initial value of tau_squared

## Estimation section: ##
# To find MLE estimates for the GPNc, use this code and results
```

```
# section:
resultsGPNC=matrix(1,nrow=km,ncol=5)
likelihood=function(theta,x){
    x=x_sim
    beta0=theta[1]
    beta1=theta[2]
    cc=2*exp(-exp(theta[3]))-1
    ssq=exp(theta[4])
    vv=ssq/(1-cc^2)
    lpdf=vector(mode="numeric",k)
    m=exp(beta0+beta1)/(1-cc)
    lpdf[1]=-.5*log(2*pi)-.5*log(vv)-((x[1]-m[1])^2/(2*vv))
    for(t in 1:q){
        lpdf[t+1]=-.5*log(2*pi)-.5*log(ssq)-((x[t+1]
            -exp(beta0+beta1*covariate[t])-cc*x[t])^2/(2*ssq))
         }
    loglike=-sum(lpdf)
    return(loglike)
}
for(i in 1:km){
    x_sim=x_sim_matrix[i,]
    ml_sim=optim(par=c(beta0_0,beta1_0,cc_0,sigmasq_0),
        likelihood,NULL,method="Nelder-Mead")
    resultsGPNc[i,]=c(ml_sim$par[1],ml_sim$par[2],
        2*exp(-exp(ml_sim$par[3]))-1,
        exp(ml_sim$par[4]),-ml_sim$val)
}
```

```
resultsGPNc
beta0_ml_sim=resultsGPNc[,1]
beta1_ml_sim=resultsGPNc[,2]
cc_ml_sim=resultsGPNc[,3]
sigmasq_ml_sim=resultsGPNc[,4]
loglike_ml_sim=resultsGPNc[,5]

# To find MLE estimates for the GSSc, use this code and results
# section:
resultsGSSc=matrix(1,nrow=km,ncol=5)
likelihood=function(theta,y){
    y=y_sim
    beta0=theta[1]
    beta1=theta[2]
    cc=2*exp(-exp(theta[3]))-1
    ssq=exp(theta[4])
    tsq=exp(theta[5])
    vv=vector(mode="numeric",k)
    m=vector(mode="numeric",k)
    lpdf=vector(mode="numeric",k)
    m[1]=exp(beta0+beta1*covariate[1])/(1-cc)
    vv[1]=(ssq/(1-cc^2))+tsq
    lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
    for(t in 1:q){
        m[t+1]=exp(beta0+beta1*covariate[t])
          +cc*(m[t]+((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
        vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
```

```
        lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
            (2*vv[t+1]))
    }
    loglike=-sum(lpdf)
    return(loglike)
}
for(i in 1:km){
    y_sim=y_sim_matrix[i,]
    ml_sim=optim(par=c(beta0_0,beta1_0,cc_0,sigmasq_0,tausq_0),
        likelihood,NULL,method="Nelder-Mead")
    resultsGSSc[i,]=c(ml_sim$par[1],ml_sim$par[2],
        2*exp(-exp(ml_sim$par[3]))-1,
        exp(ml_sim$par[4]),exp(ml_sim$par[5]))
}

resultsGSSc

beta0_ml_sim=resultsGSSc[,1]; beta1_ml_sim=resultsGSSc[,2]
cc_ml_sim=resultsGSSc[,3]; sigmasq_ml_sim=resultsGSSc[,4]
tausq_ml_sim=resultsGSSc[,5]
```

Appendix B: Additional Figures

We have included additional figures not included in the main body of the manuscript for any researchers wishing for more information about what happens in certain situations, especially in the case of increasing observation error while modeling with both the GPNc and GSSc.

Appendix B.1. GPNc Figures

GPNc Simulation Results without Observation Error:

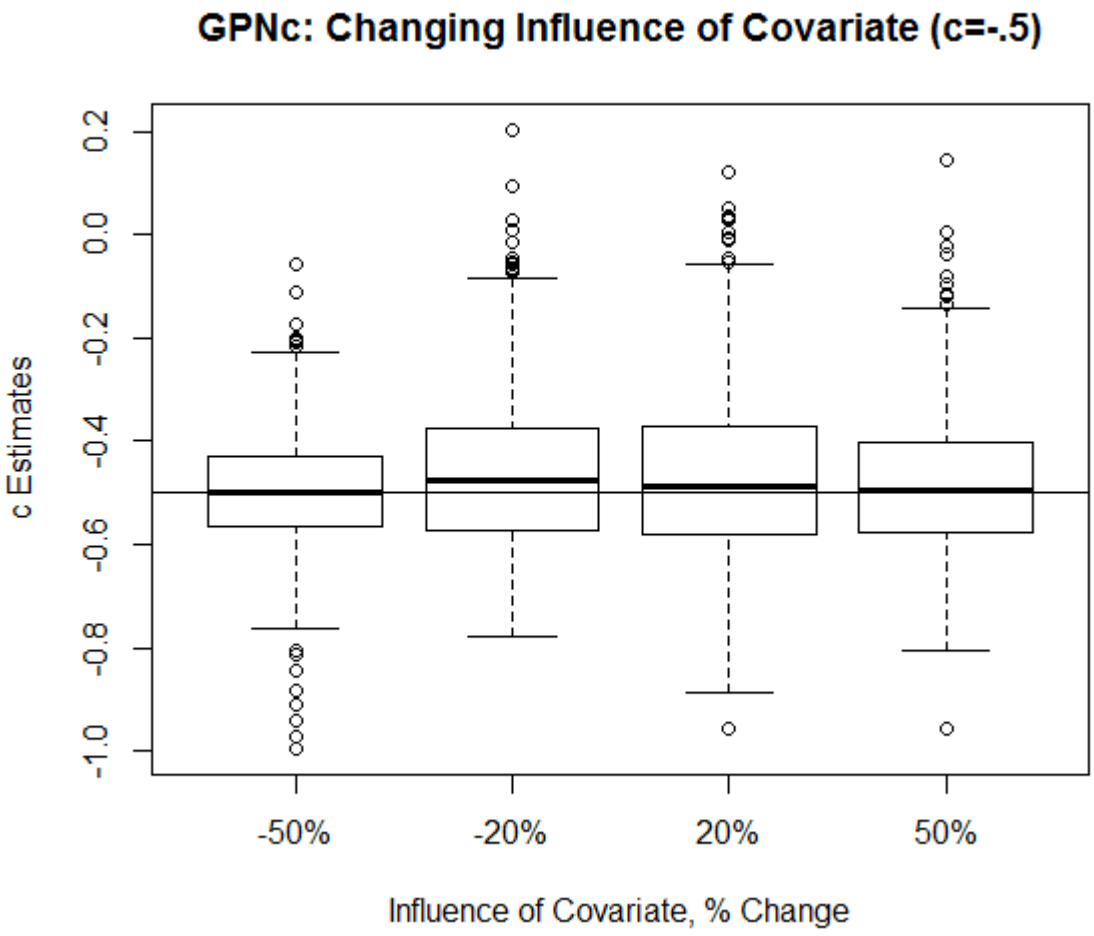


Fig B.1.1 As the influence of the covariate changes from a -50% stationary mean population change with a single standard deviation change in the covariate to a -20% change, a 20% change, and a 50% change, there is convergence around the true value of c.

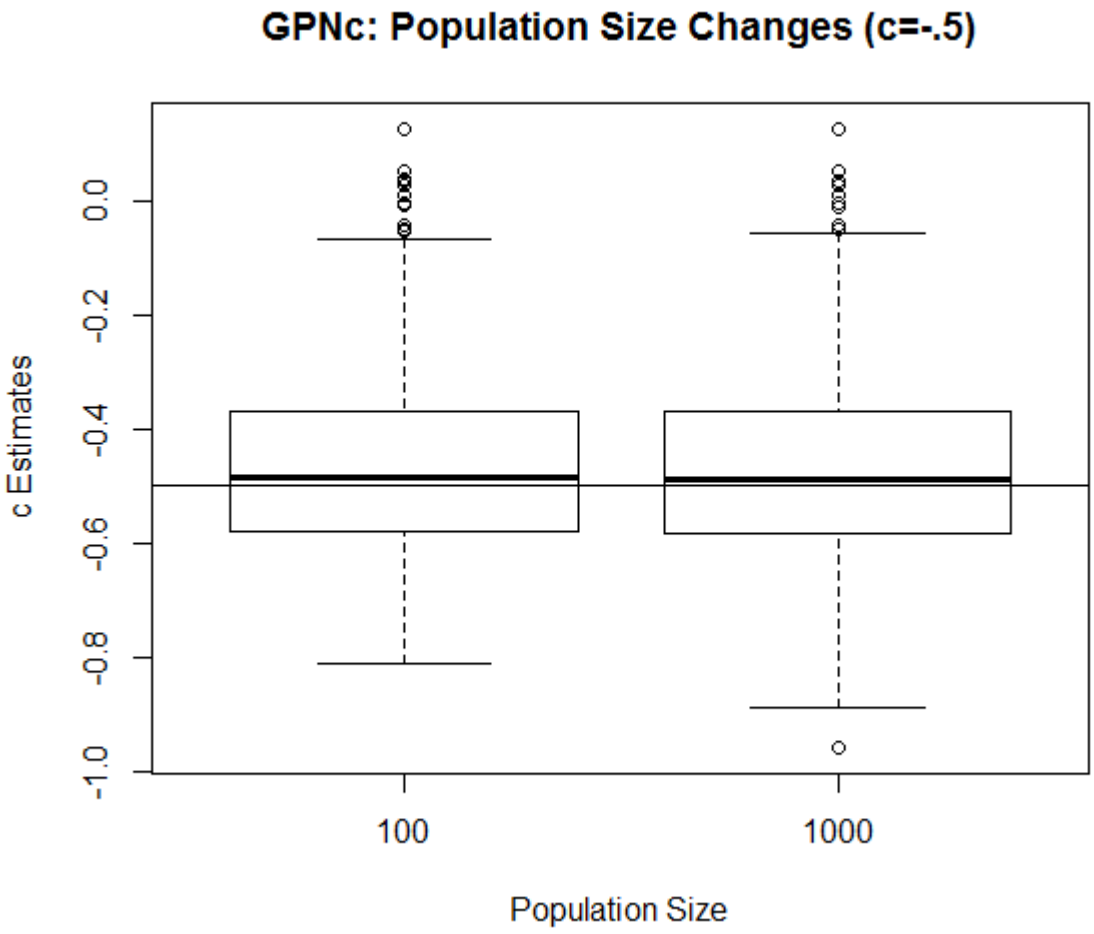


Fig B.1.2. As the population size changes from 100 to 1000, we find no clear distinction in variance or ability of the model to converge on the actual value of c. Both converge well.

GPNc Simulation Results with Unmodeled Observation Error:

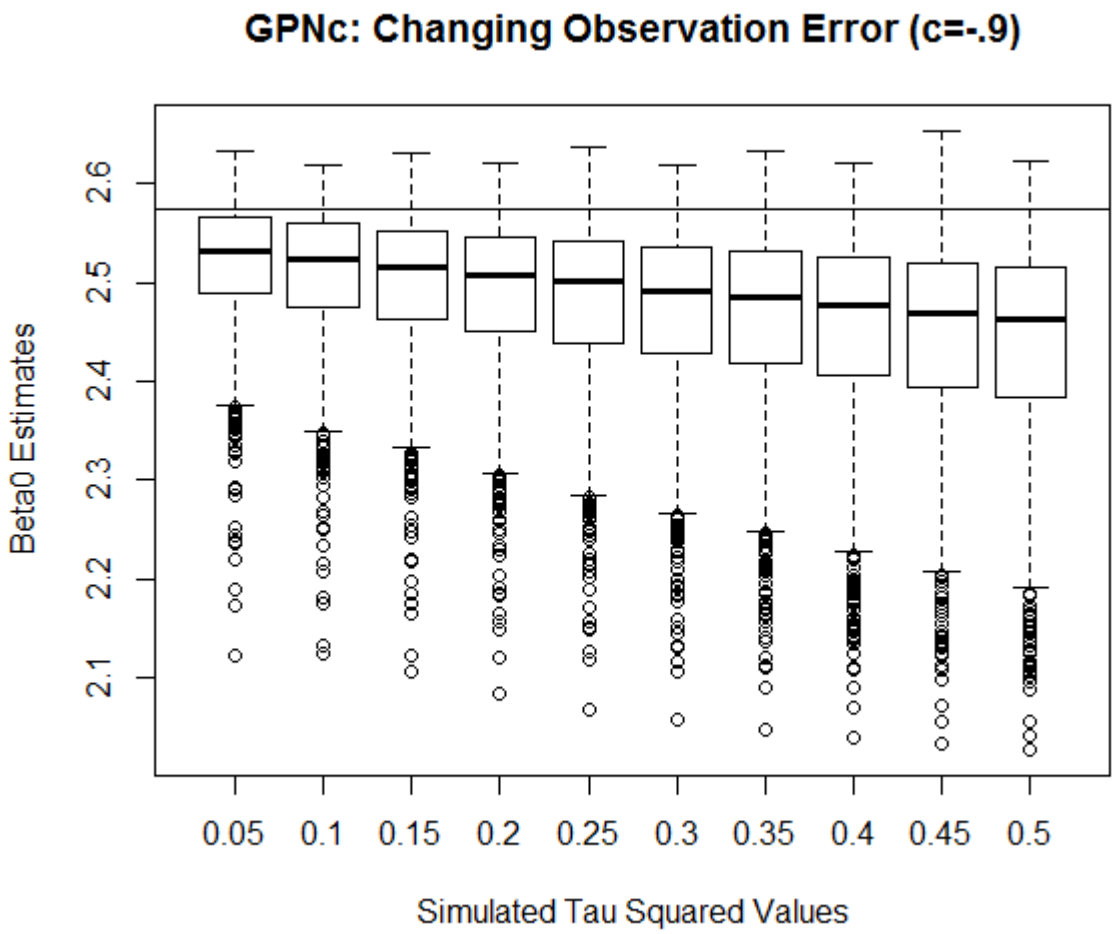


Fig B.1.3. When unmodeled observation error is added to the GPNc with$c = -0.9$ in 0.05 increments from 0.05 to 0.5, we see increasing bias and variance in the β_0 estimates.

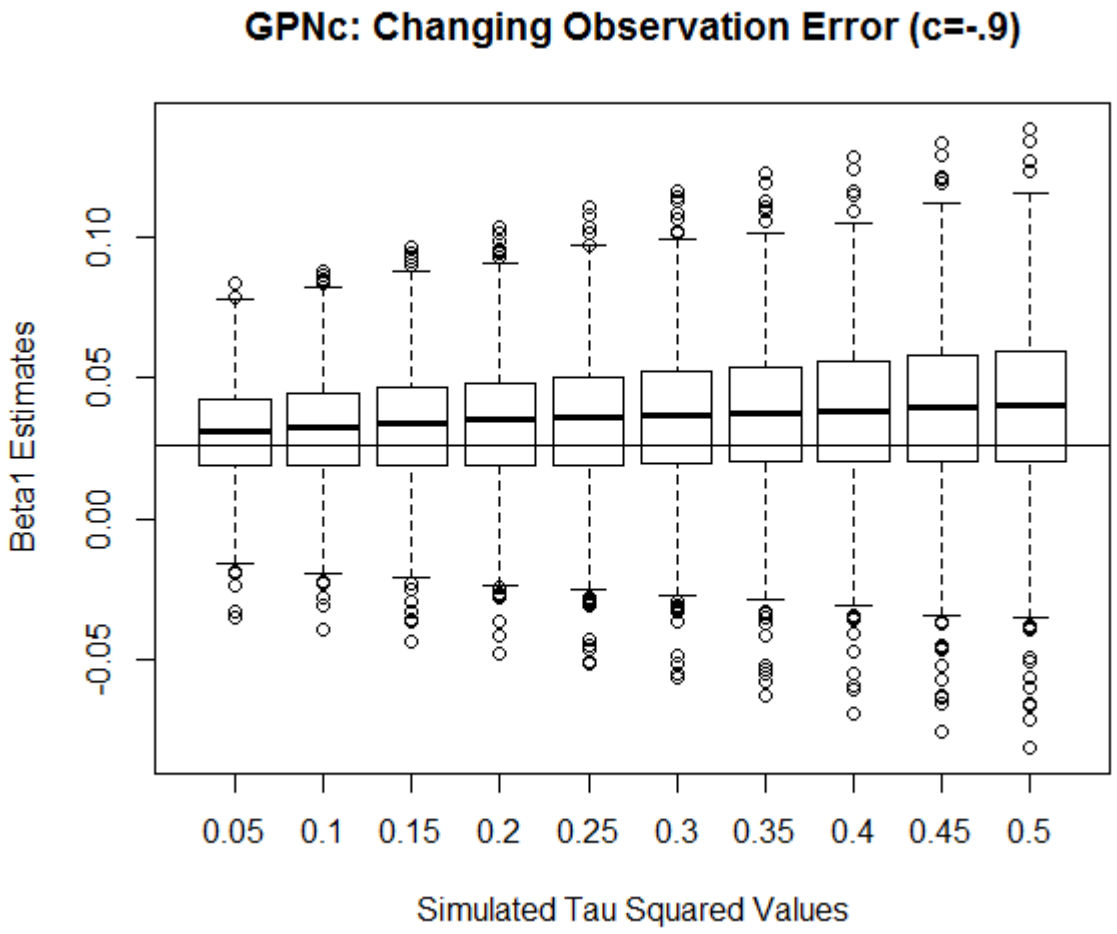


Fig B.1.4. When unmodeled observation error is added to the GPNc with$c = -0.9$ in 0.05 increments from 0.05 to 0.5, we see increasing bias and variance in the β_1 estimates.

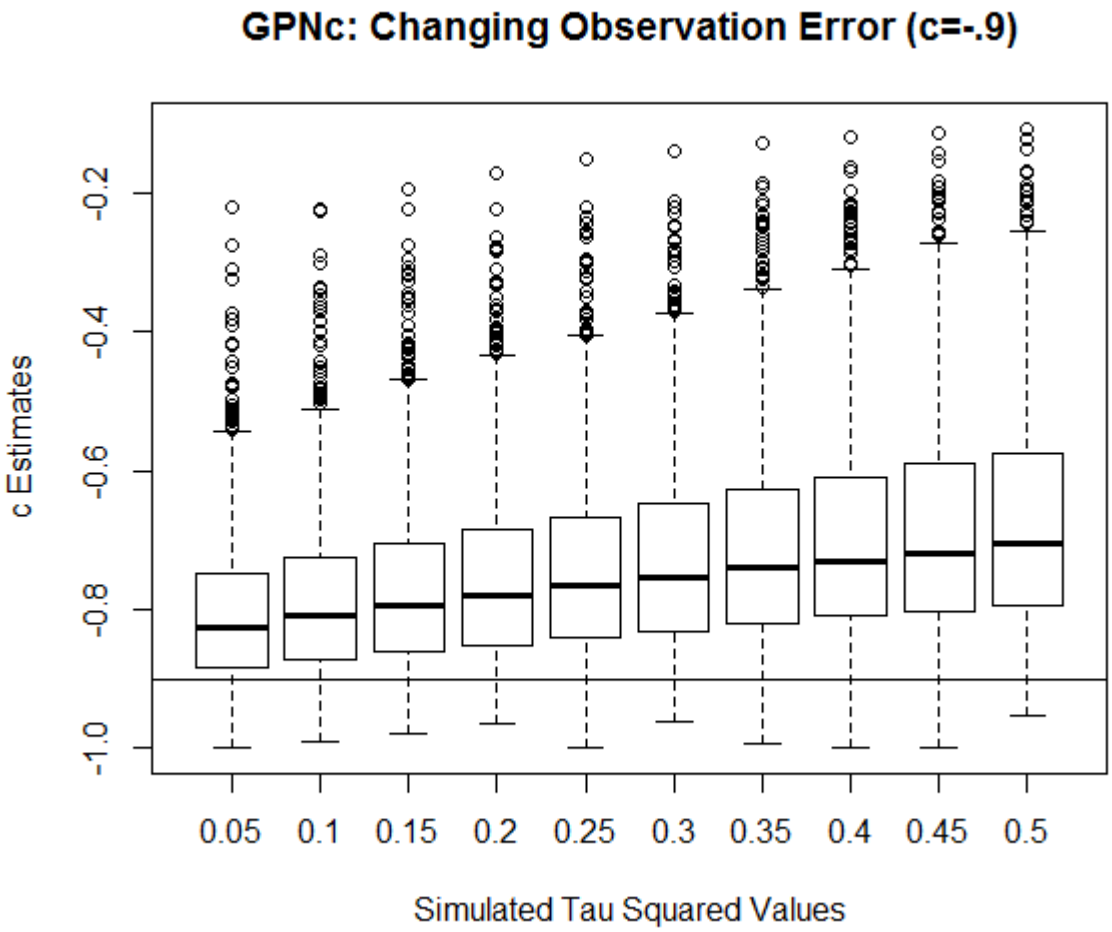


Fig B.1.5. When unmodeled observation error is added to the GPNc with $c = -0.9$ in 0.05 increments from 0.05 to 0.5, we see increasing bias and variance in the c estimates.

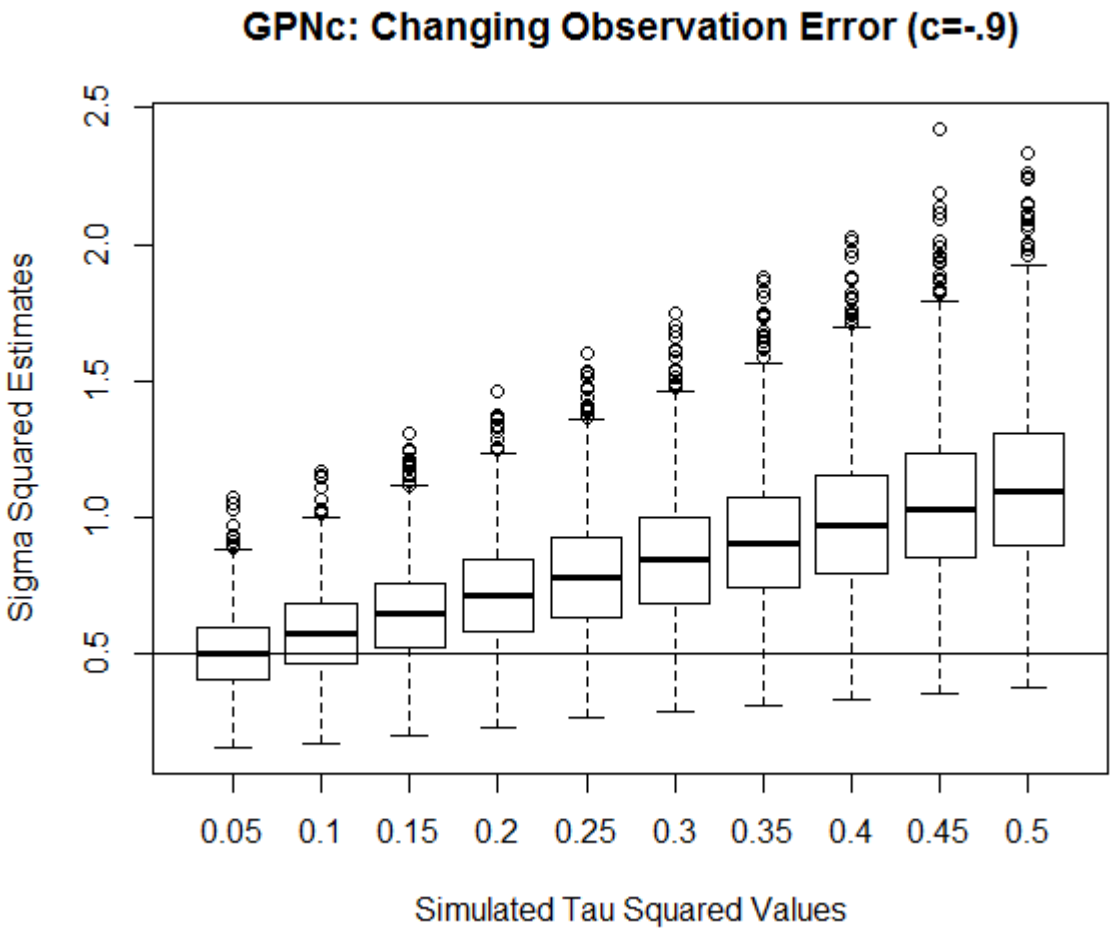


Fig B.1.6. When unmodeled observation error is added to the GPNc with $c = -0.9$ in 0.05 increments from 0.05 to 0.5, we see greatly increasing bias and variance in the σ^2 estimates.

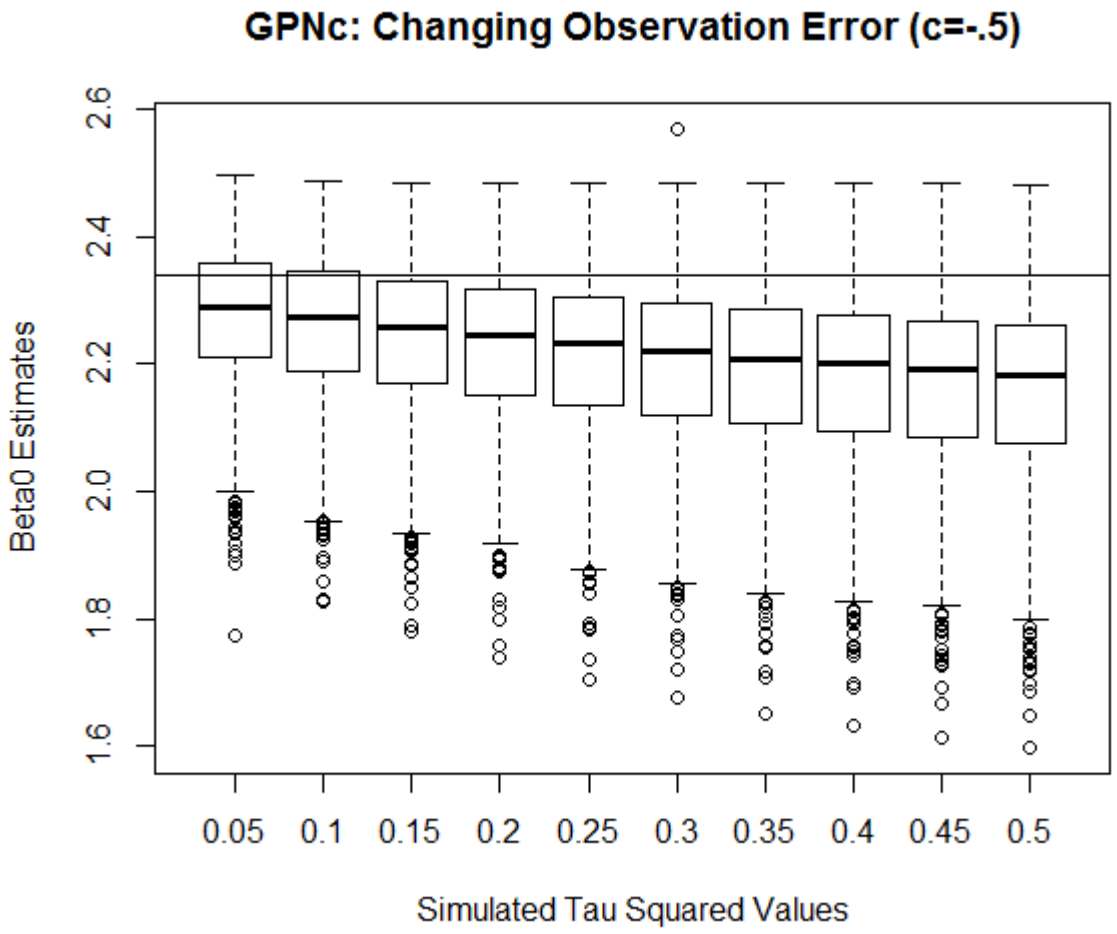


Fig B.1.7. When unmodeled observation error is added to the GPNc with $c = -0.5$ in 0.05 increments from 0.05 to 0.5, we see increasing bias and variance in the β_0 estimates.

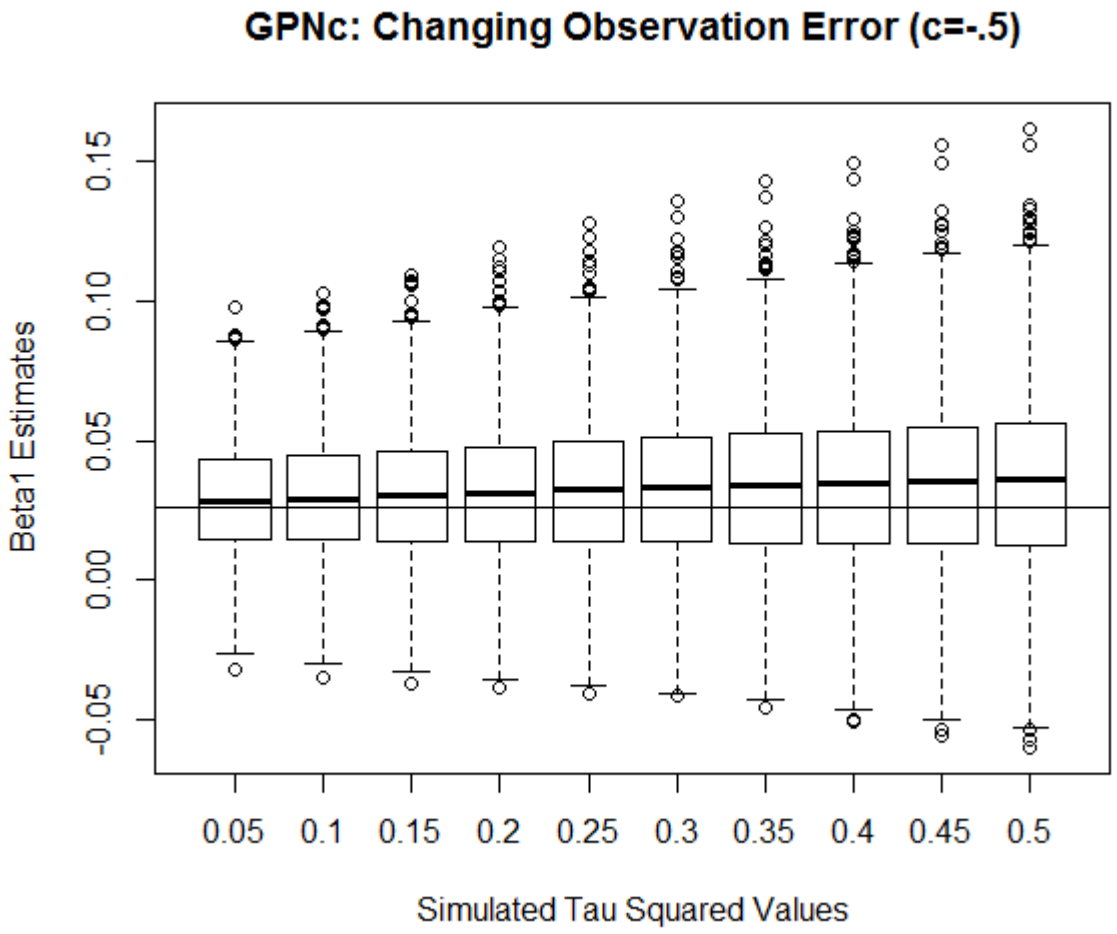


Fig B.1.8. When unmodeled observation error is added to the GPNc with $c = -0.5$ in 0.05 increments from 0.05 to 0.5, we see increasing variance in the β_1 estimates, though less increasing bias than in other circumstances.

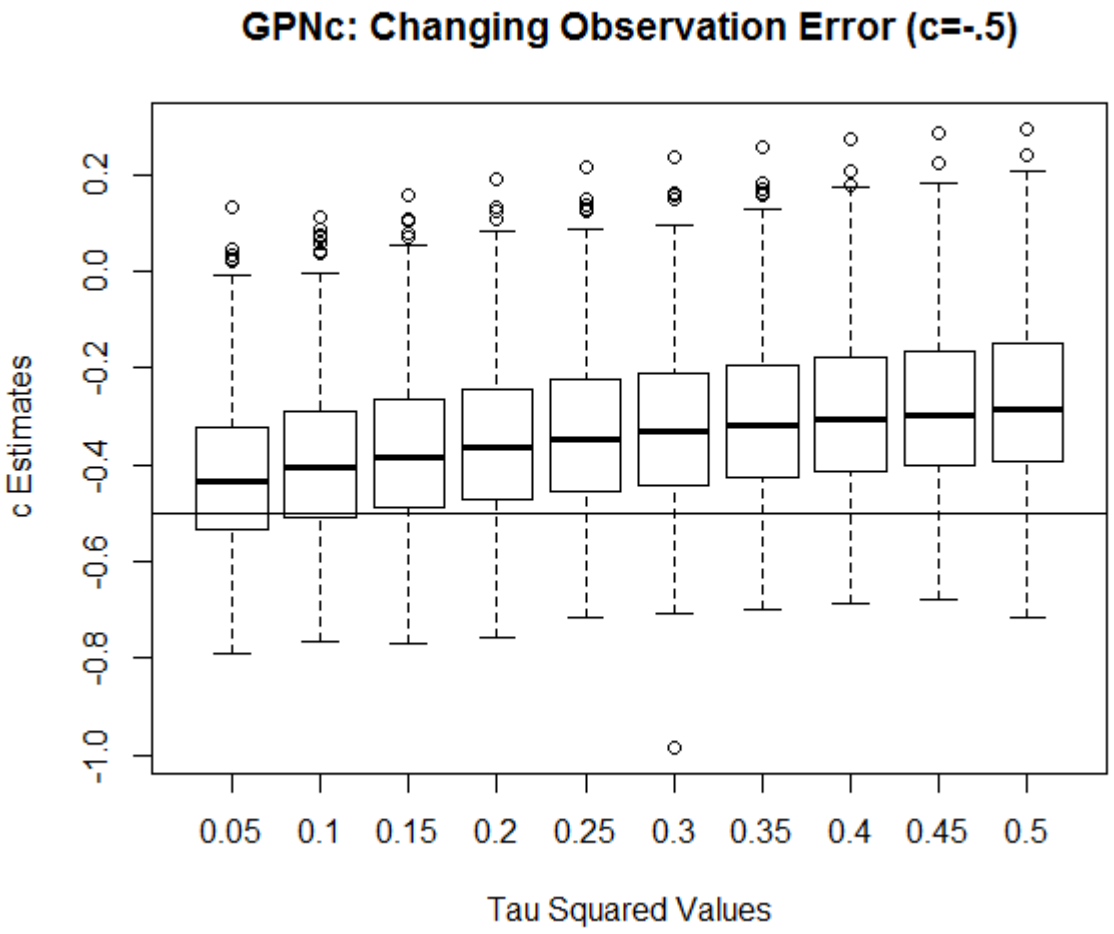


Fig B.1.9. When unmodeled observation error is added to the GPNc with $c = -0.5$ in 0.05 increments from 0.05 to 0.5, we see increasing bias in the c estimates. Variance remains similar throughout.

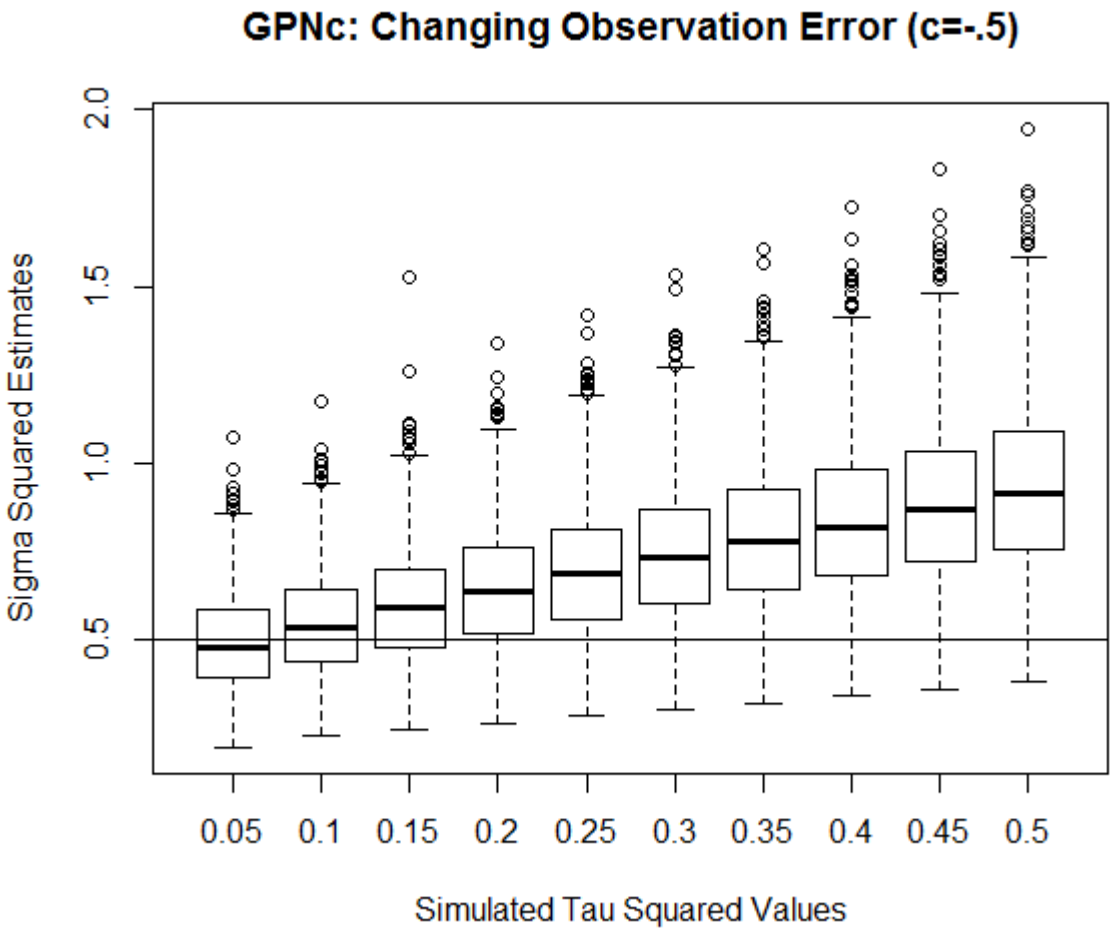
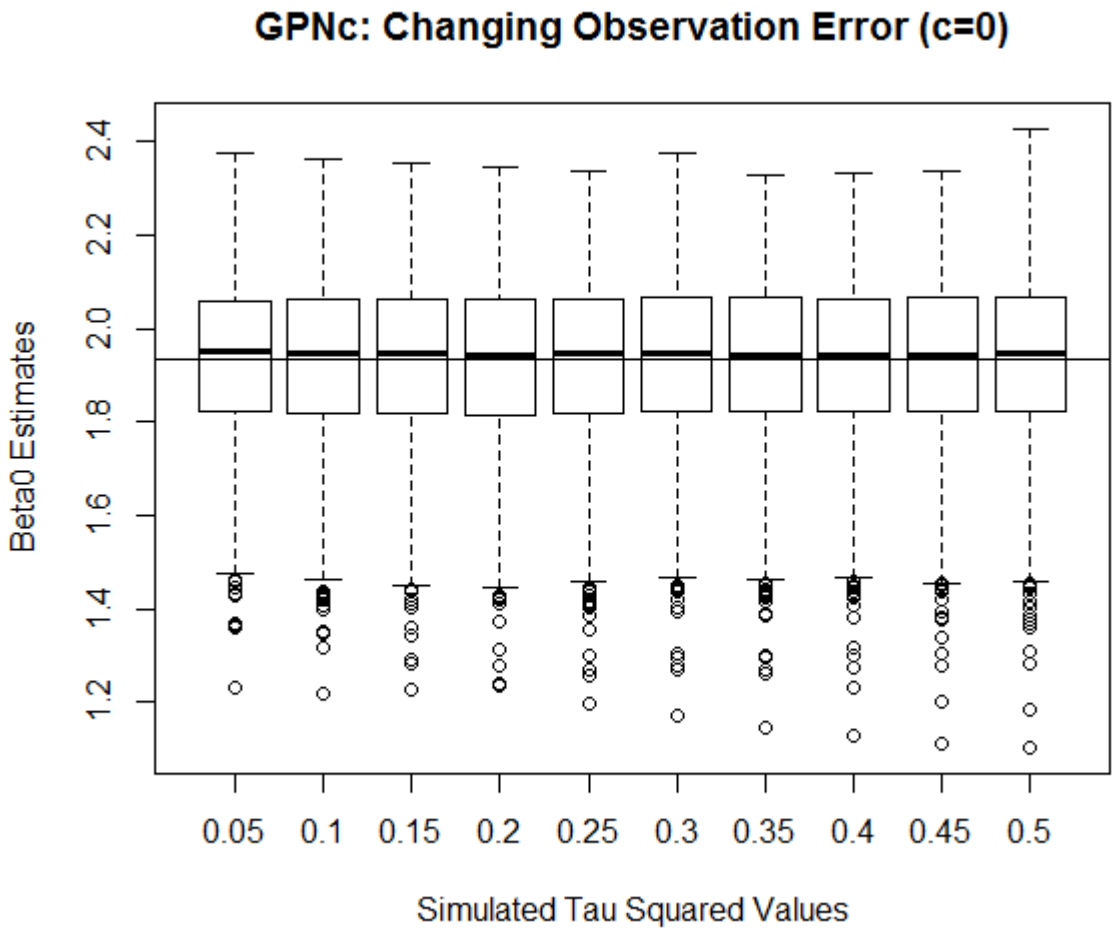


Fig B.1.10. When unmodeled observation error is added to the GPNc with $c = -0.5$ in 0.05 increments from 0.05 to 0.5, we see increasing bias and variance in the σ^2 estimates.



Fig

B.1.11. When unmodeled observation error is added to the GPNc with $c = 0$ in 0.05 increments from 0.05 to 0.5, we see no bias and the same variance through estimates of β_0. The case where $c = 0$ is clearly the best-case scenario for situations with unmodeled observation error.

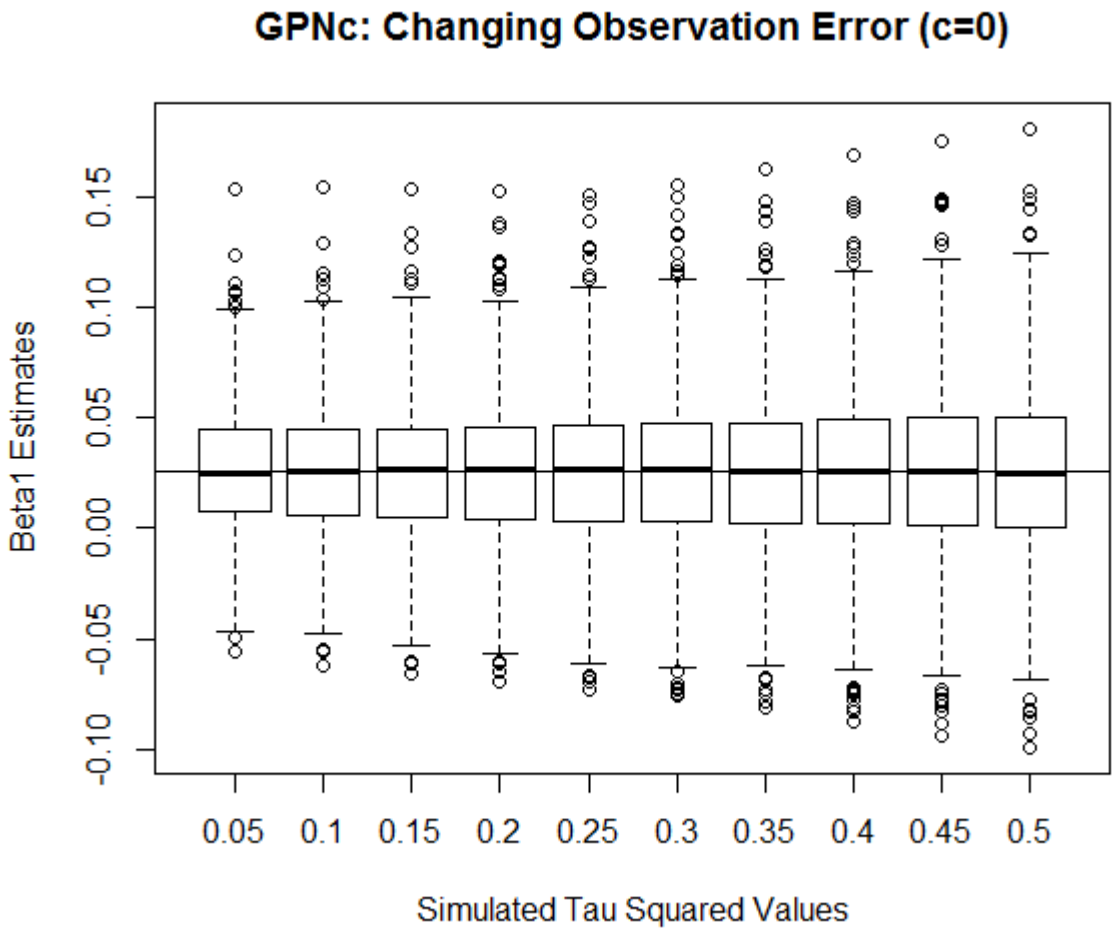


Fig B.1.12. When unmodeled observation error is added to the GPNc with $c = 0$ in 0.05 increments from 0.05 to 0.5, we see no bias and only slightly increasing variance through estimates of β_1. The case where $c = 0$ is clearly the best-case scenario for situations with unmodeled observation error.

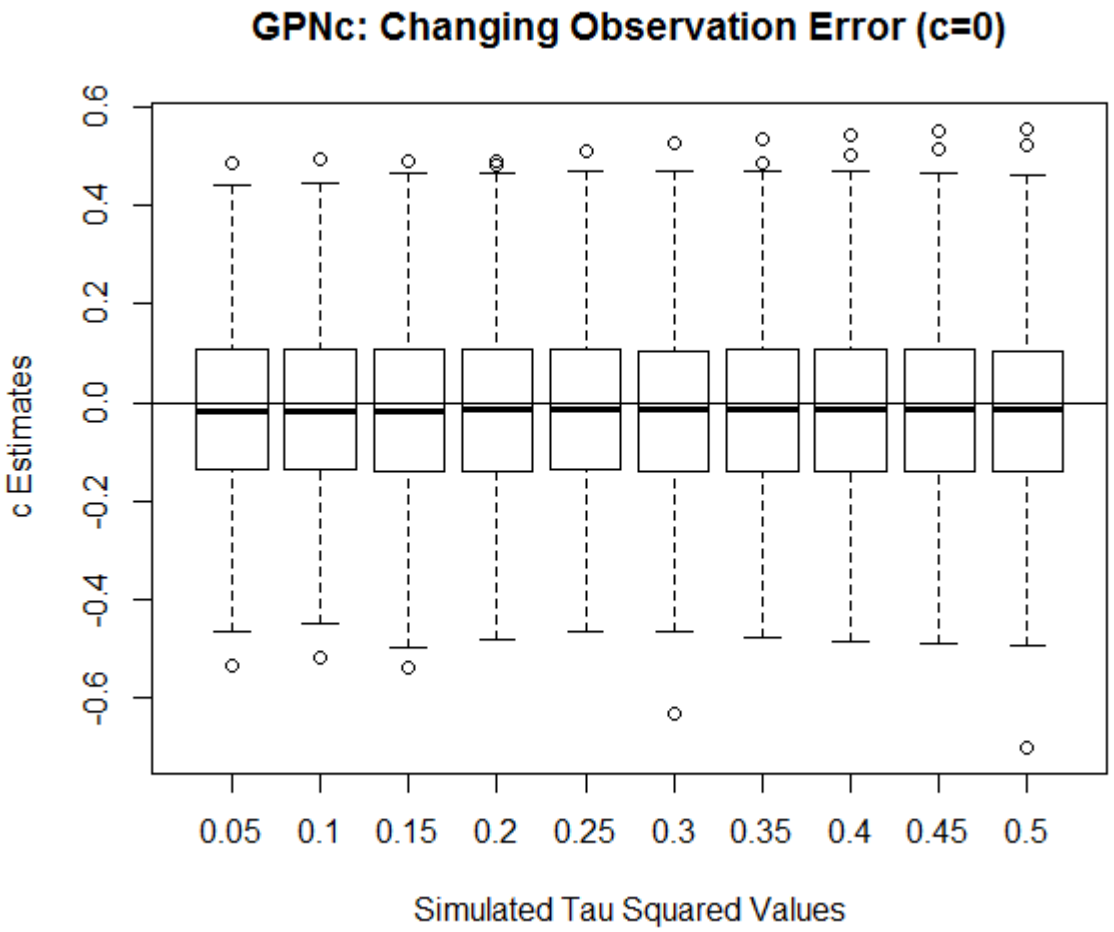


Fig B.1.13. When unmodeled observation error is added to the GPNc with $c = 0$ in 0.05 increments from 0.05 to 0.5, we see no bias and no change in variance through estimates of c. The case where $c = 0$ is clearly the best-case scenario for situations with unmodeled observation error.

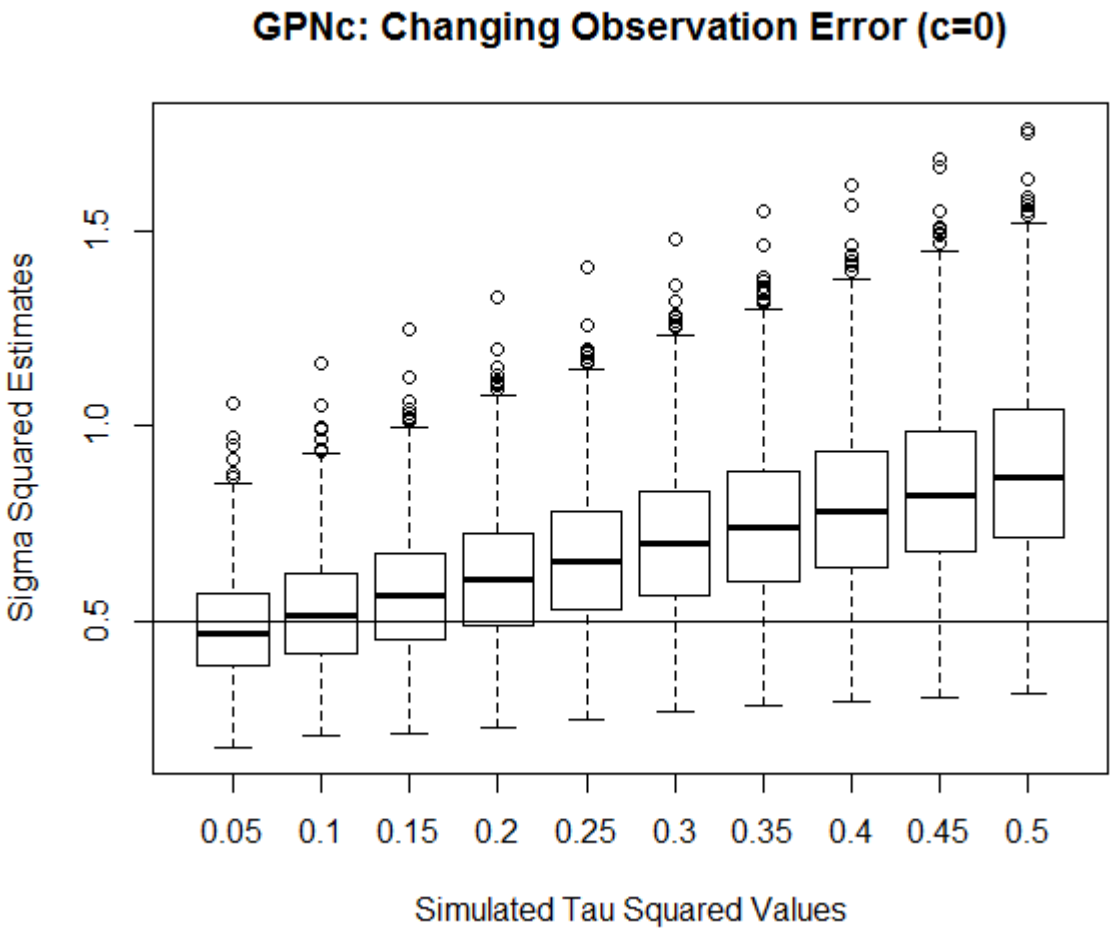


Fig B.1.14. When unmodeled observation error is added to the GPNc with $c = 0$ in 0.05 increments from 0.05 to 0.5, we see greatly increasing bias and variance through estimates of σ^2. The case where $c = 0$ is clearly the best-case scenario for situations with unmodeled observation error as ssq has taken most of the additional error.

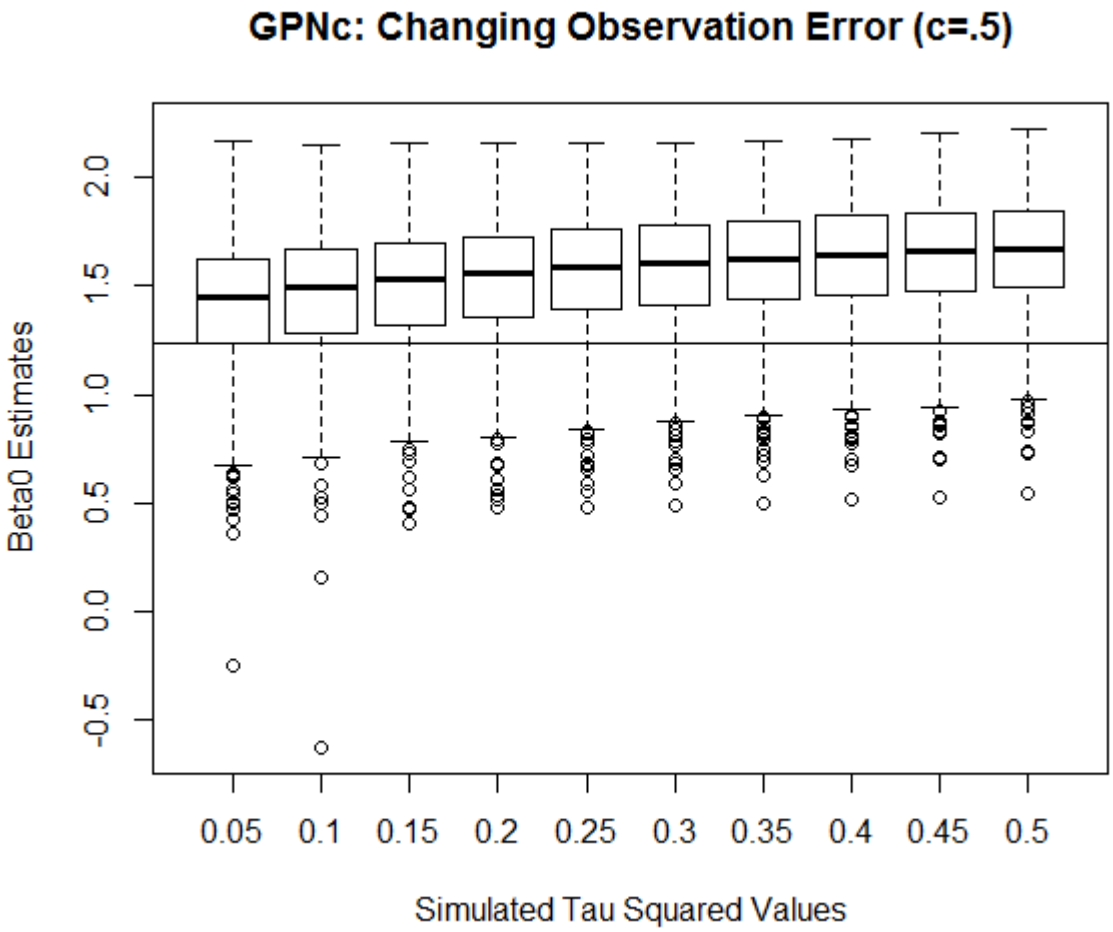


Fig B.1.15. When unmodeled observation error is added to the GPNc with $c = 0.5$ in 0.05 increments from 0.05 to 0.5, we see increasing bias in the *beta*0 estimates. Variance remains about the same throughout.

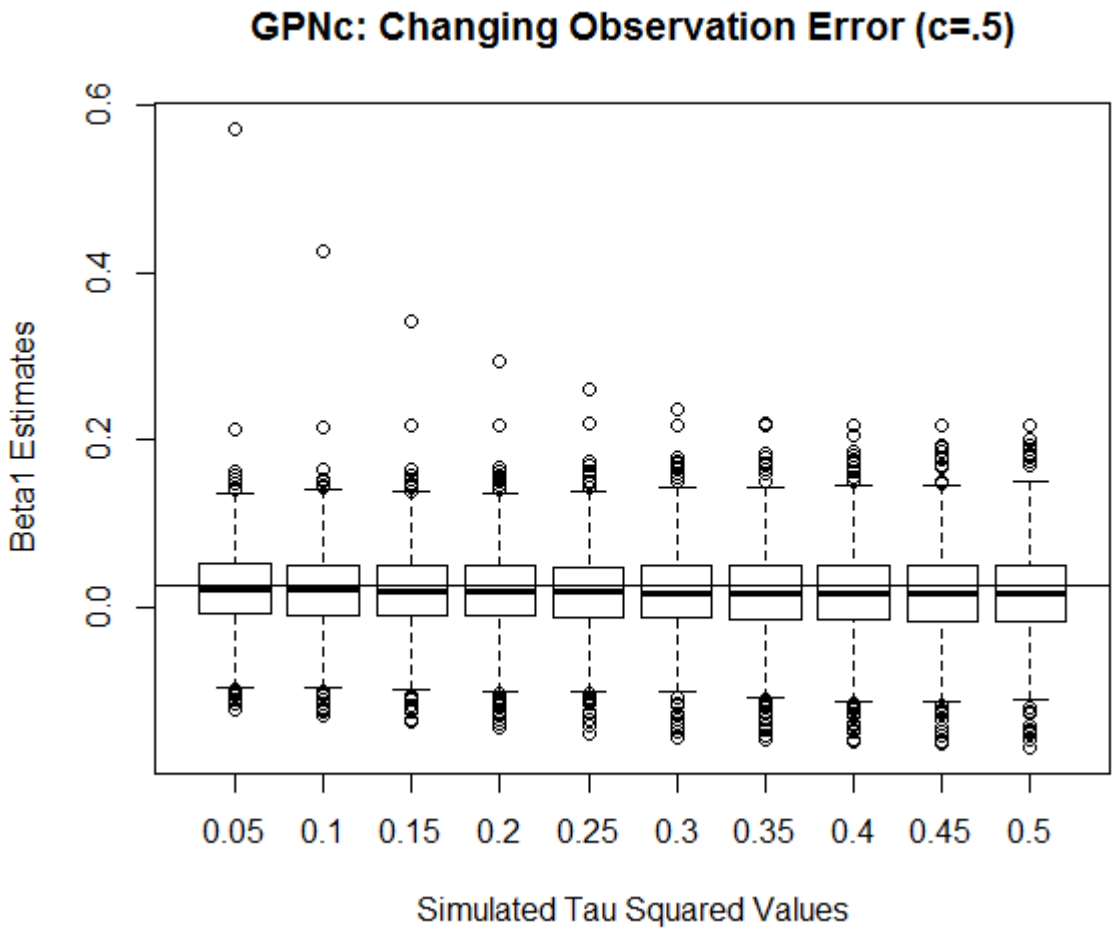


Fig B.1.16. When unmodeled observation error is added to the GPNc with $c = 0.5$ in 0.05 increments from 0.05 to 0.5, we see minimal bias and no real change in variance in the β_1 estimates.

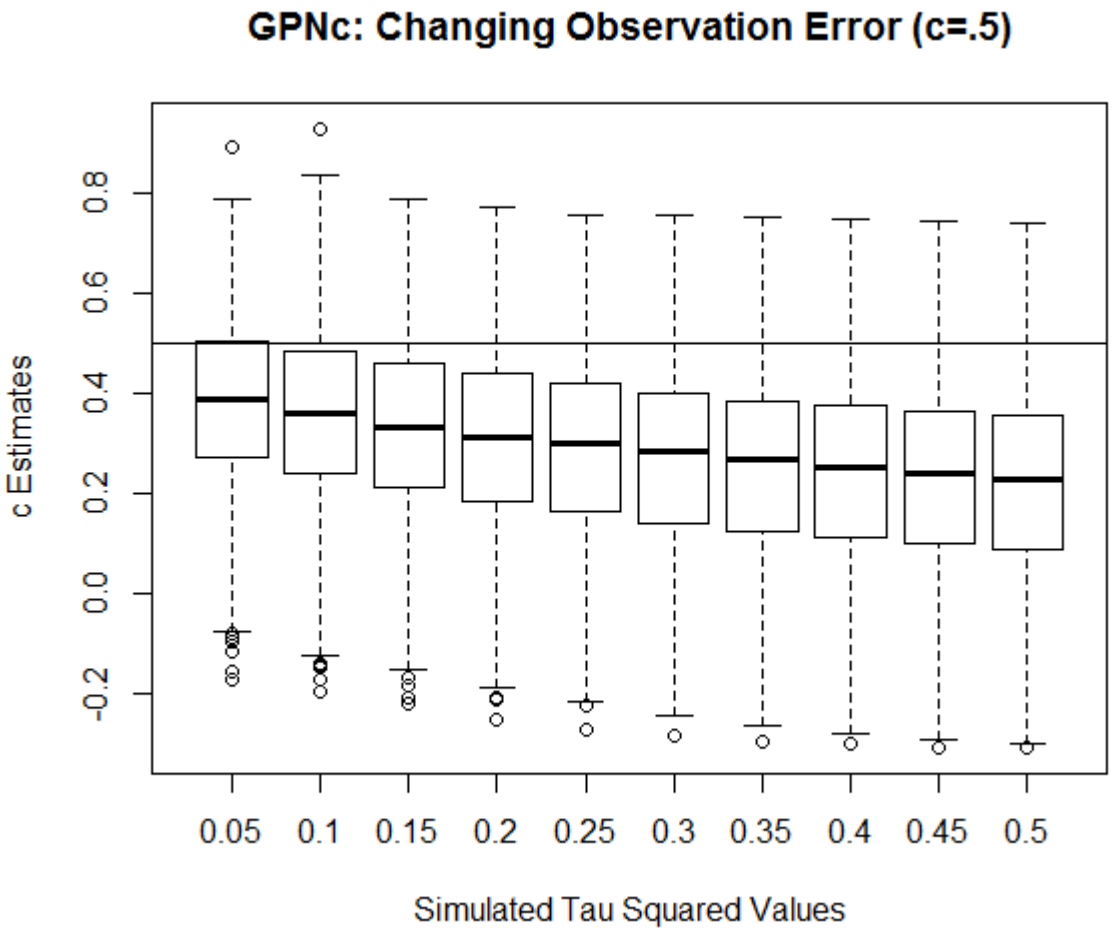


Fig B.1.17. When unmodeled observation error is added to the GPNc with $c = 0.5$ in 0.05 increments from 0.05 to 0.5, we see increasing bias, though similar variance, in the c estimates.

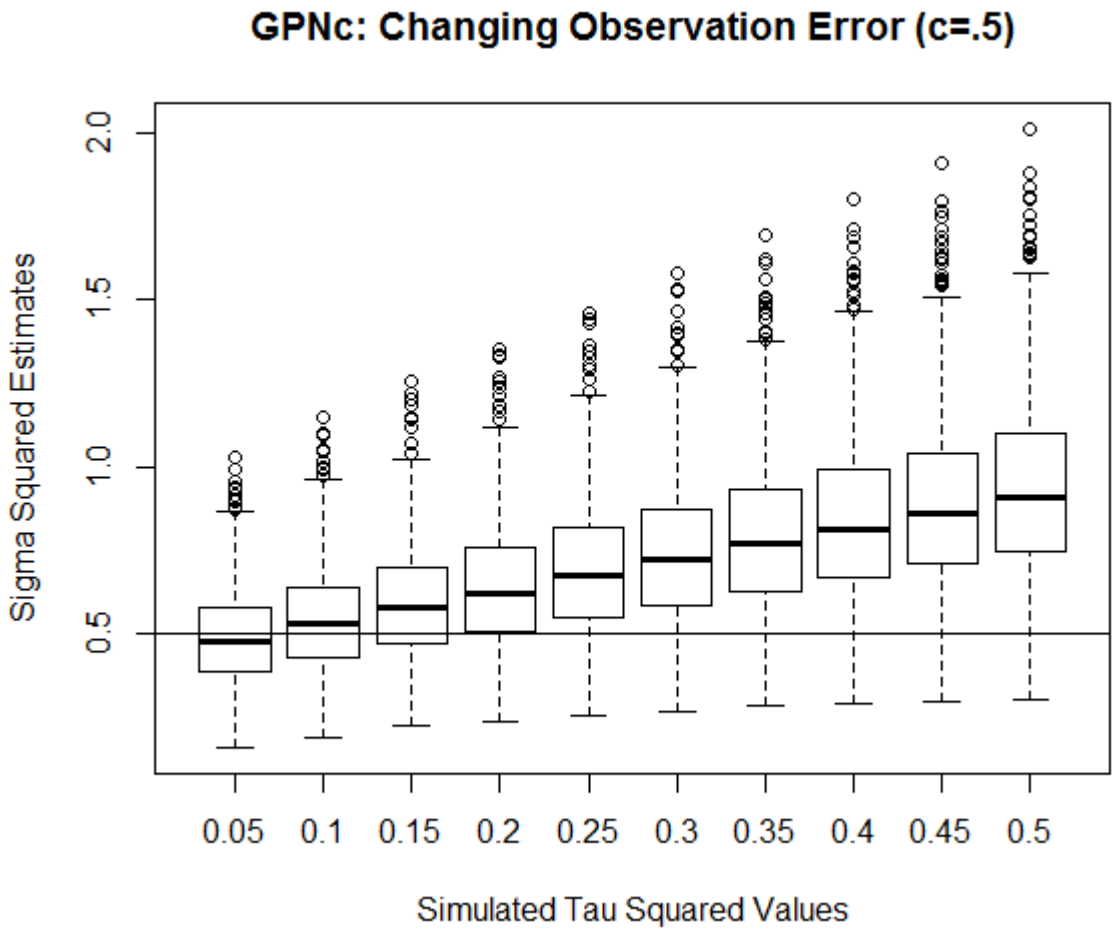


Fig B.1.18. When unmodeled observation error is added to the GPNc with $c = 0.5$ in 0.05 increments from 0.05 to 0.5, we see increasing bias and variance in the σ^2 estimates.

Graphs for β_0, β_1, and σ^2 estimates with unmodeled observation error when $c = .9$ are not shown here because they include many extreme values. When $c = .9$, the GPNc model should not be used. A graph for c, below, is shown as reference for the large estimability problems.

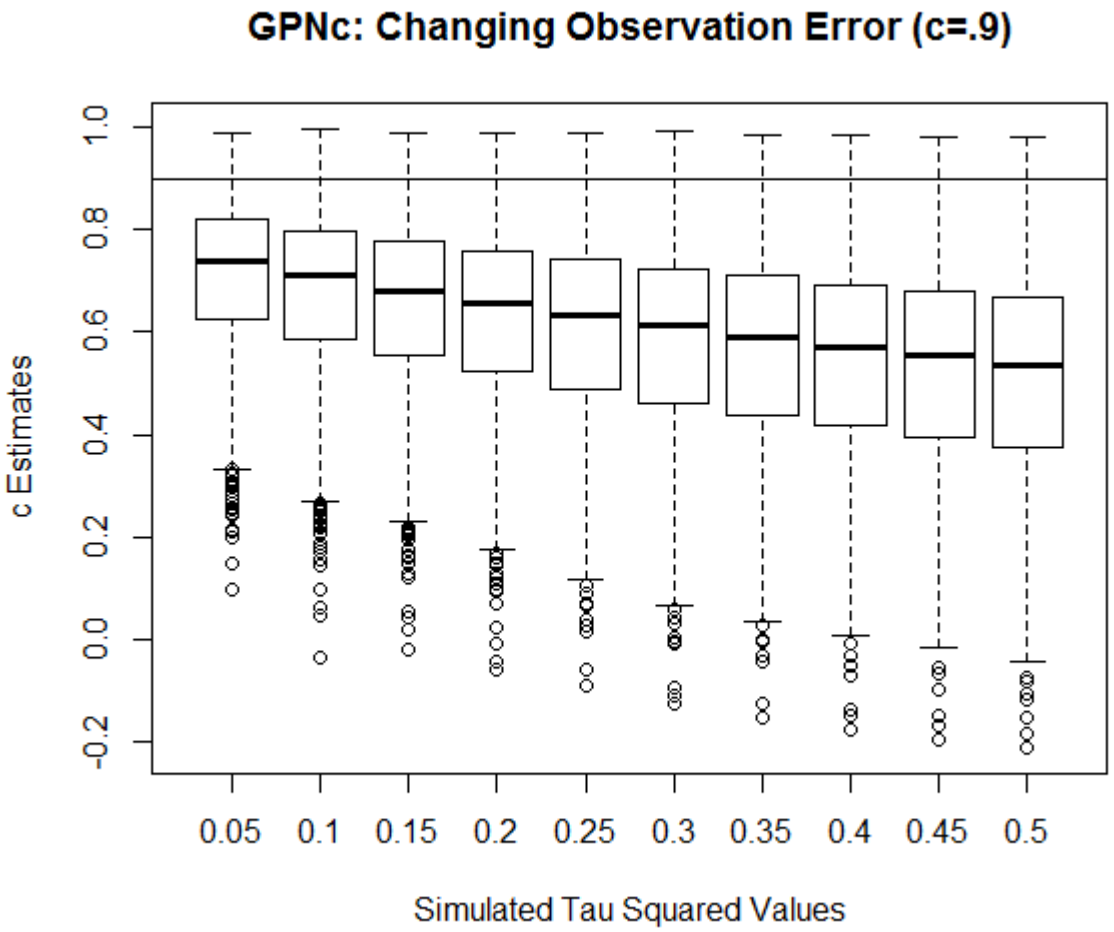


Fig B.1.19. When unmodeled observation error is added to the GPNc with $c = 0.9$ in 0.05 increments from 0.05 to 0.5, we see greatly increasing bias and variance in the c estimates.

Appendix B.2. GSSc Figures

GSSc Simulation Results:

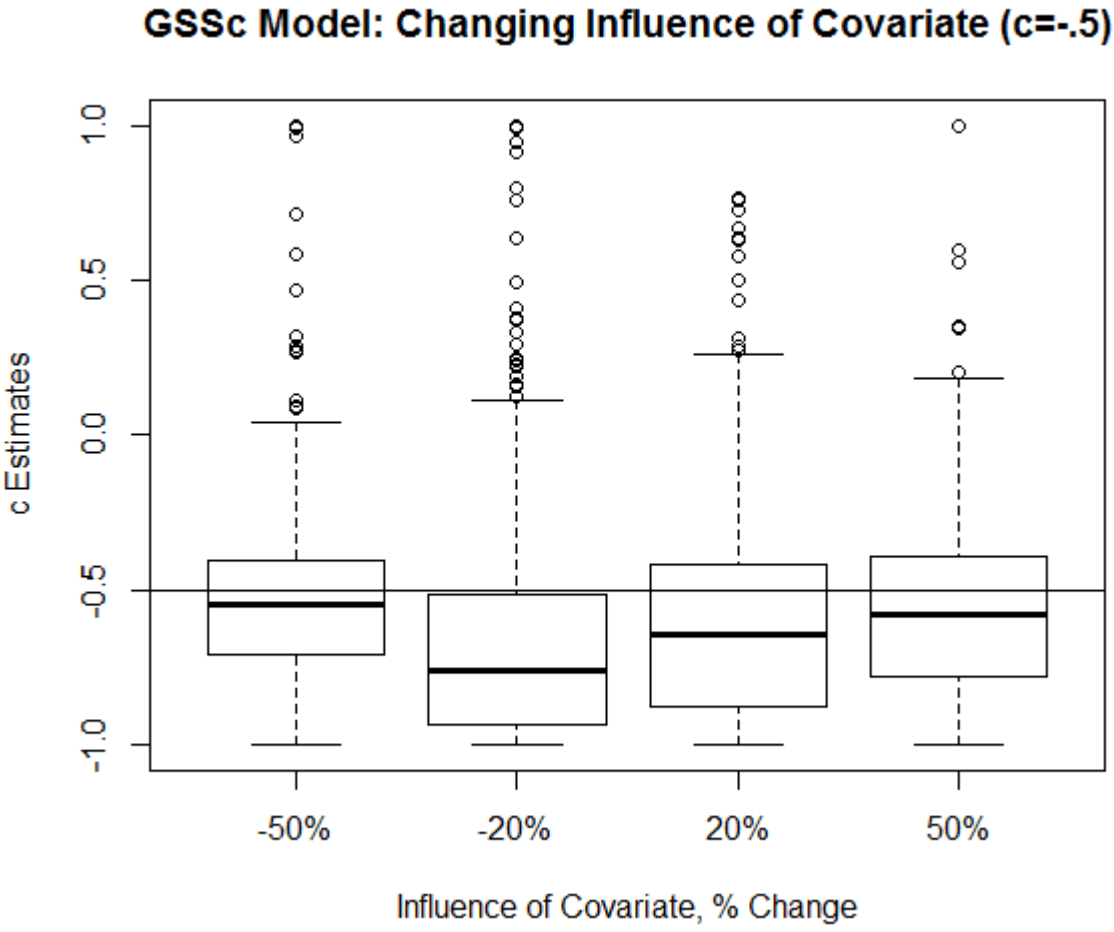


Fig B.2.1. Changing the influence of a single standard deviation change of the covariate shows that there is slightly less variance as well as less bias as influence increases in absolute value..

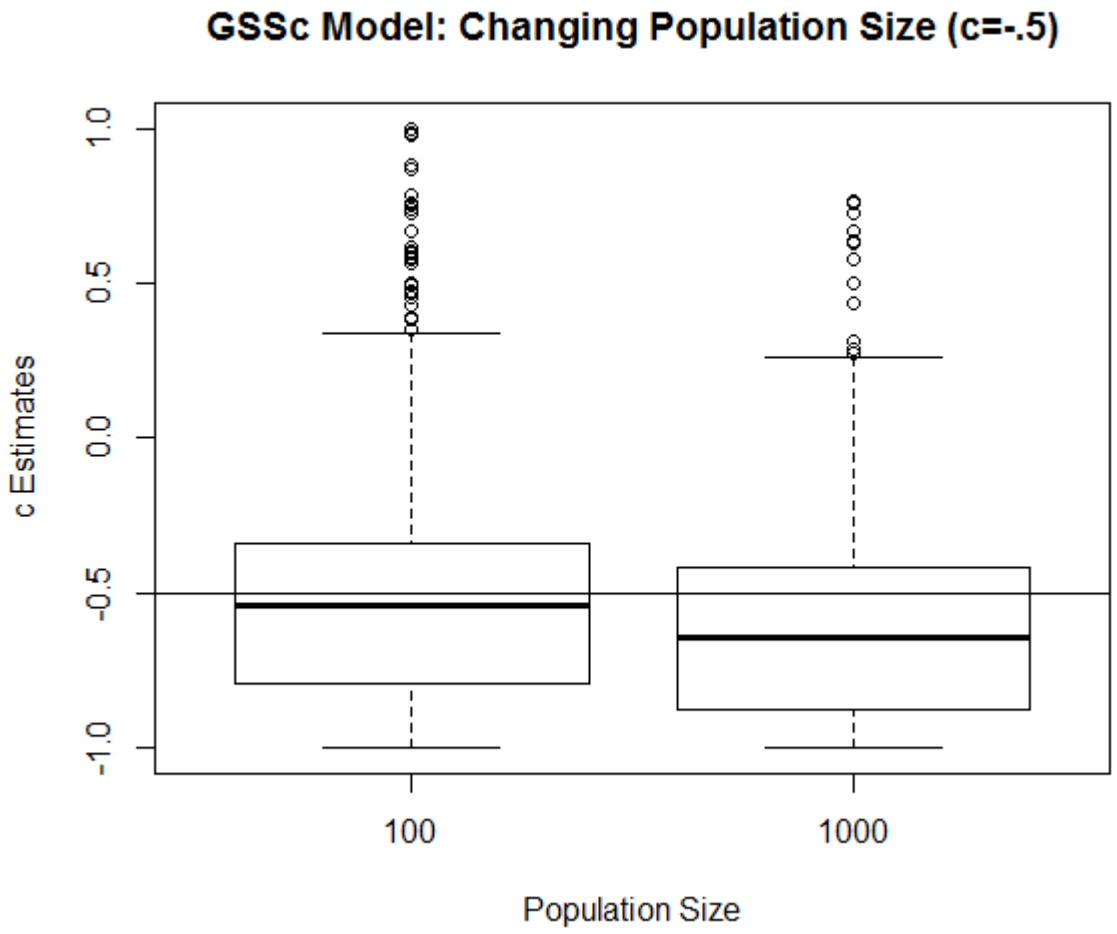


Fig B.2.2. Changing the stationary mean of the population results in similar variance of the estimates, but more extreme values in the smaller population size. However, there is slightly more bias in the larger population size.

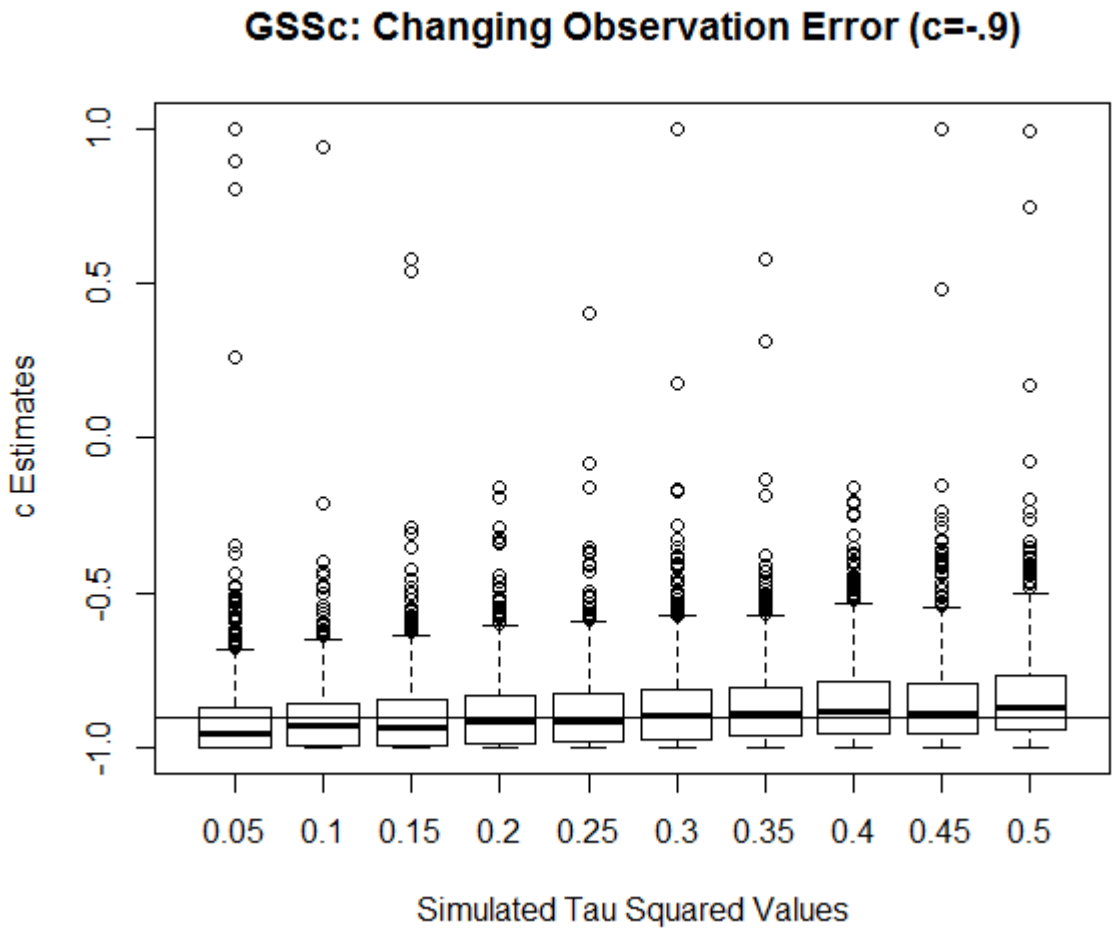


Fig B.2.3. As observation error is added to the GSSc when $c = -.9$ in 0.05 increments from 0.05 to 0.5, there are extreme estimates of c throughout, though there is relatively little bias.

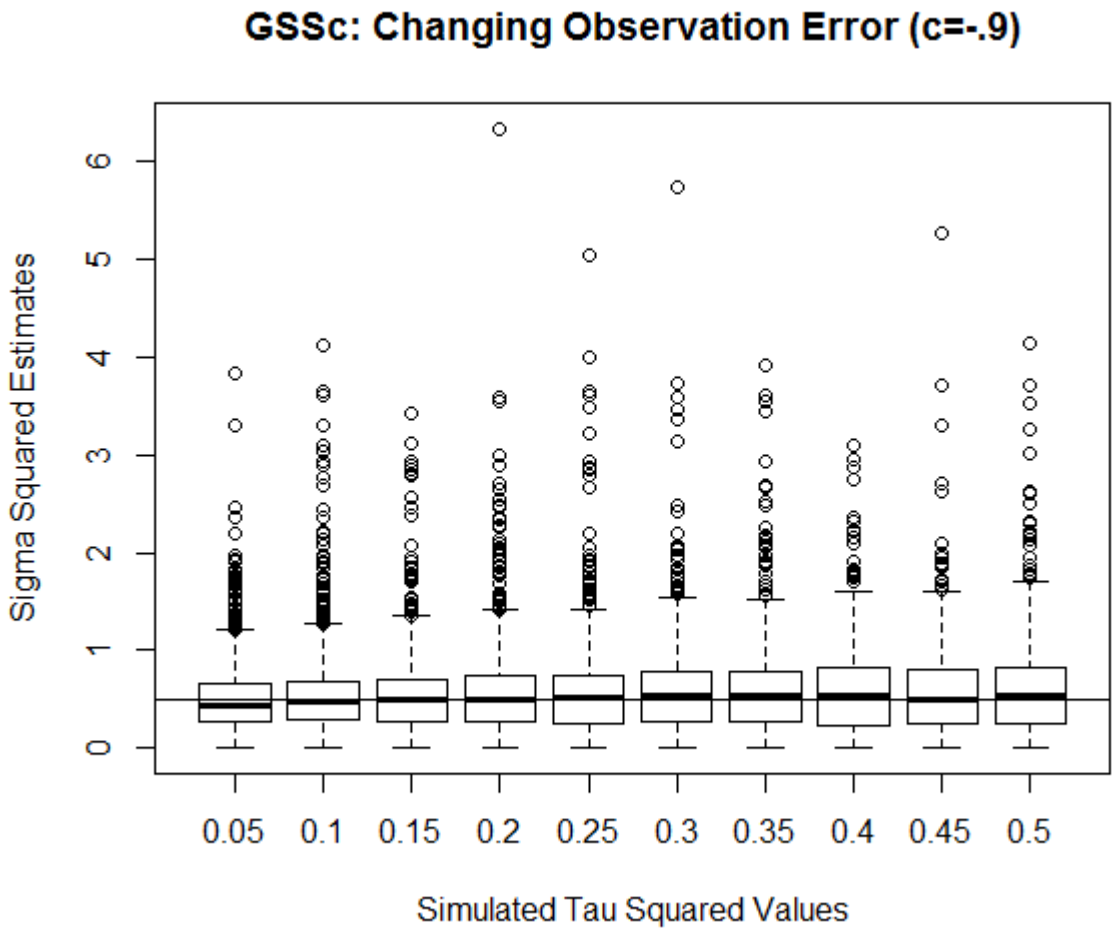


Fig B.2.4. As observation error is added to the GSSc when $c = -.9$ in 0.05 increments from 0.05 to 0.5, there are extreme estimates of σ^2 throughout, though there is relatively little bias.

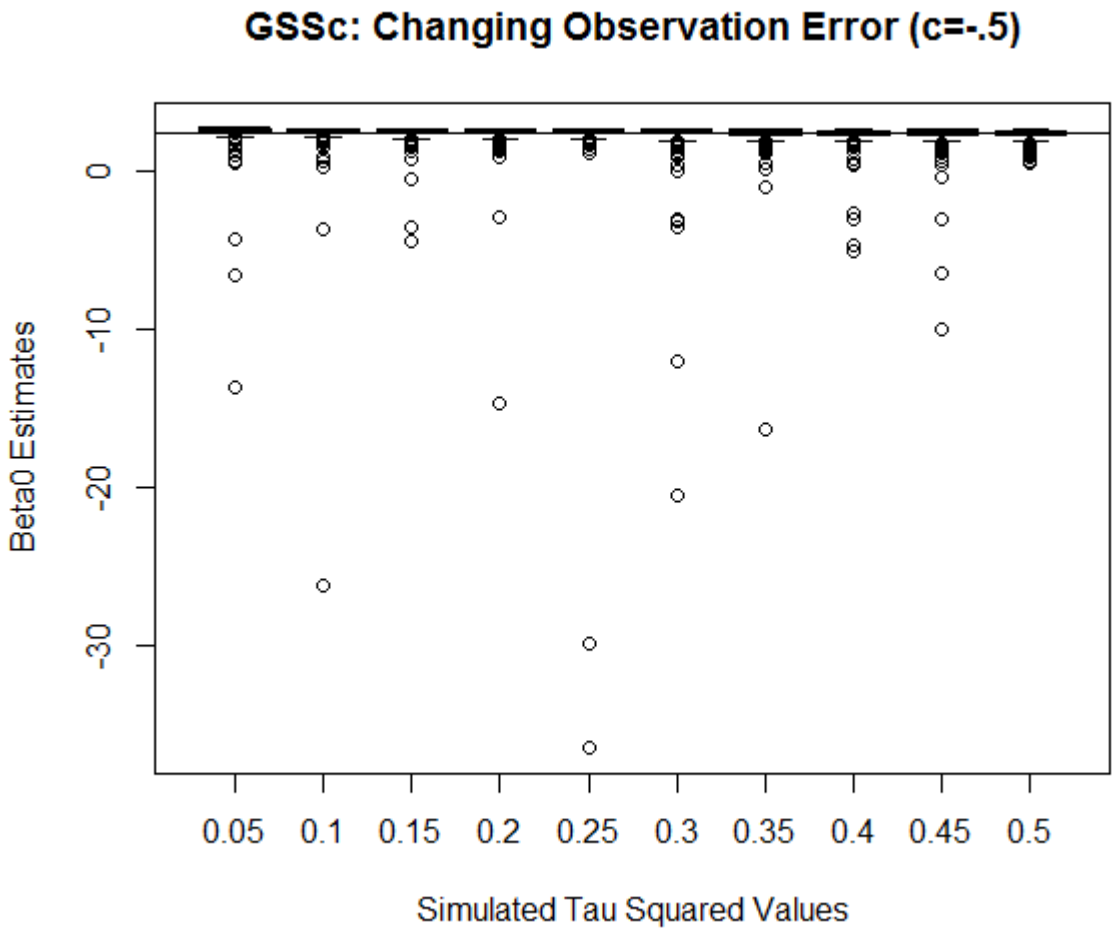


Fig B.2.5.(a) As observation error is added to the GSSc when $c = -.5$ in 0.05 increments from 0.05 to 0.5, we see such extreme estimates for β_0 that the majority of estimates are all but hidden. The real value of β_0 here is 2.34, but estimates range all the way down to nearly -40.

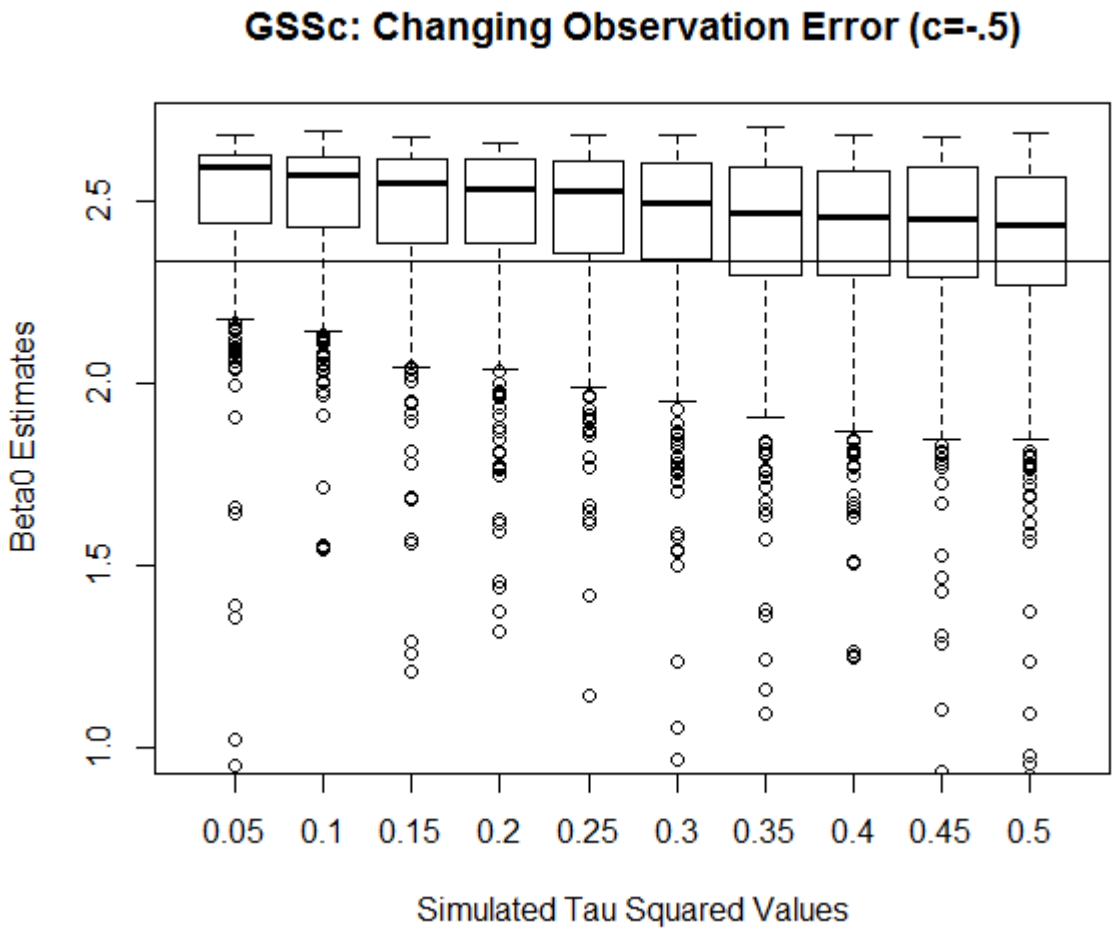


Fig B.2.5.(b) A "magnified" version of Fig B.2.5.(a). As observation error is added to the GSSc when $c = -.5$ in 0.05 increments from 0.05 to 0.5, we see such extreme estimates for β_0 that in order to see how estimates relate to the real value, we had to cut off the graph at 1.0. Even excluding the extreme values, we see quite a bit of bias and increasing variance in our β_0 estimates.

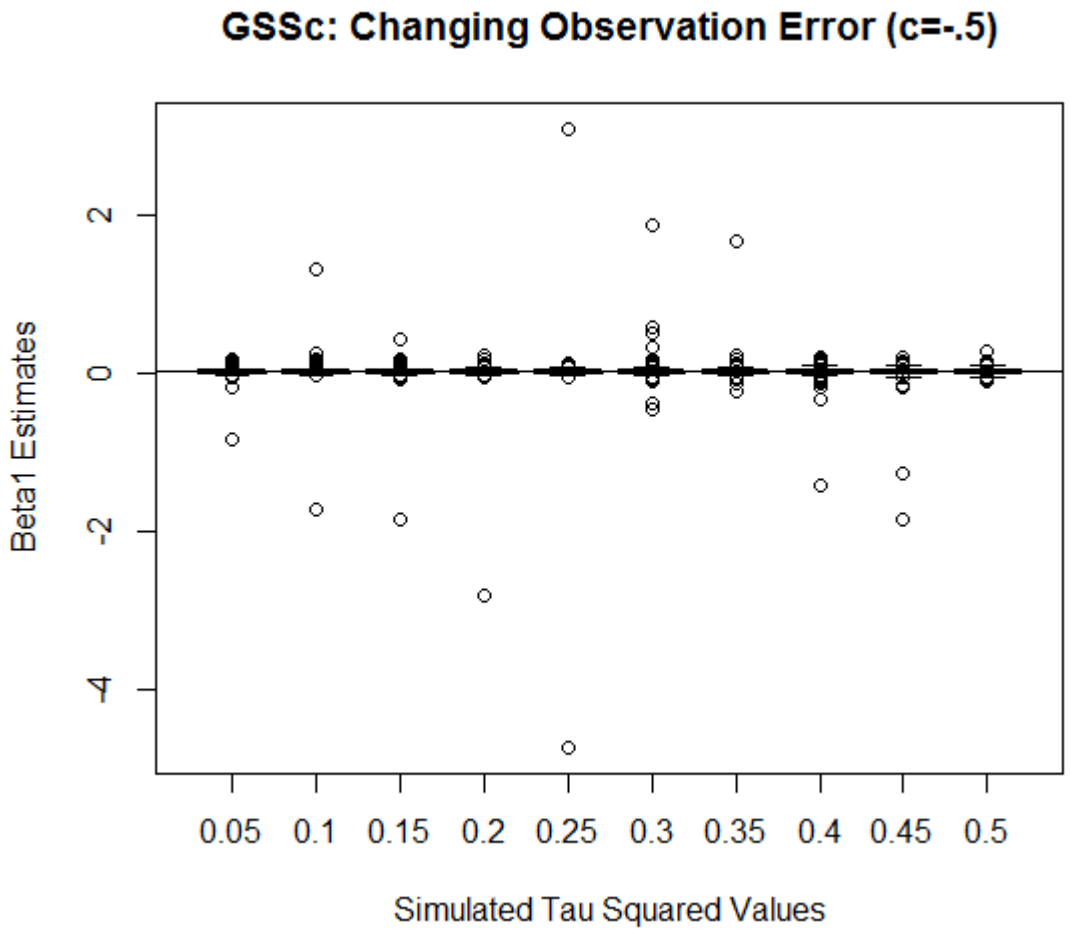


Fig B.2.6.(a) As observation error is added to the GSSc when $c = -.5$ in 0.05 increments from 0.05 to 0.5, we see such extreme estimates for β_1 that the majority of estimates are all but hidden. The real value of β_1 here is 0.03, but estimates range all the way from nearly 3 to nearly -5.

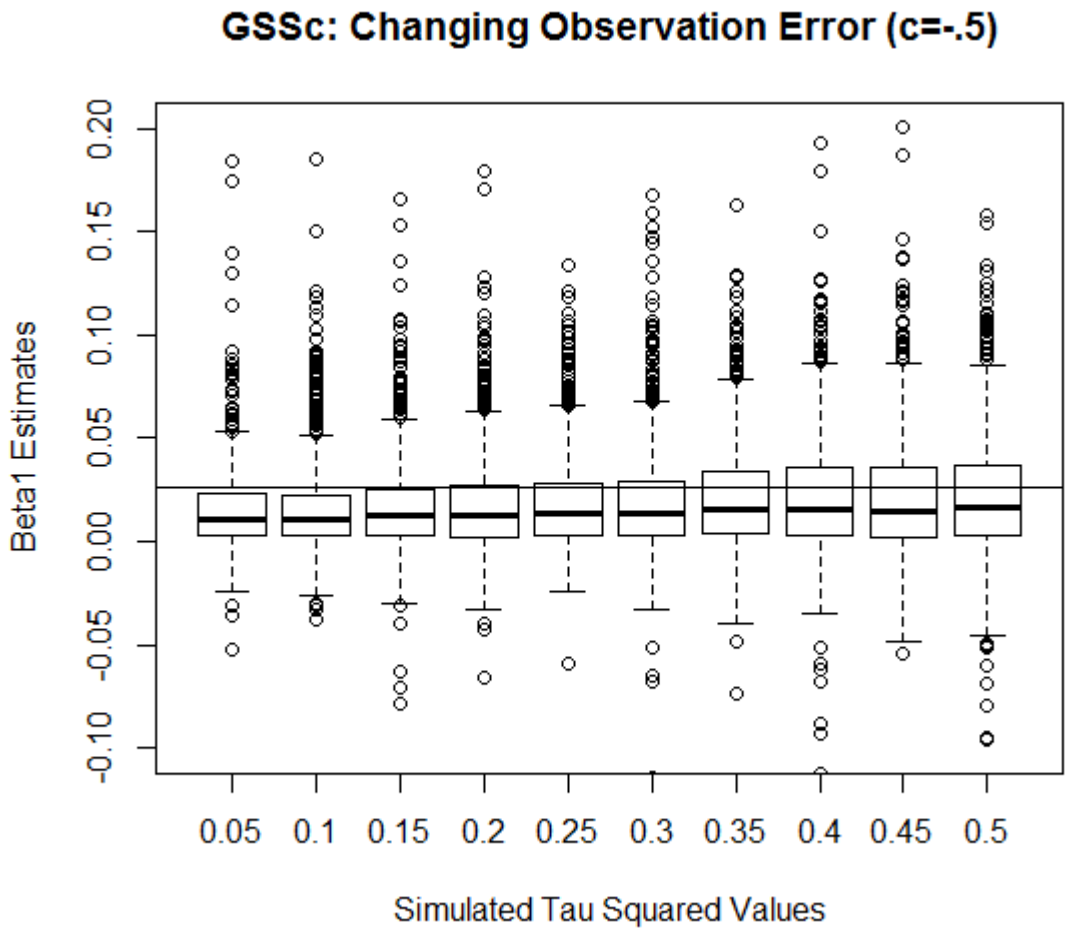


Fig B.2.6.(b) A "magnified" version of Fig B.2.6.(a). As observation error is added to the GSSc when $c = -.5$ in 0.05 increments from 0.05 to 0.5, we see such extreme estimates for β_1 that in order to see how estimates relate to the real value, we had to cut off the graph at 0.2 and -0.2. Excluding the extreme values, we do not see much bias and increasing variance in our β_1 estimates.

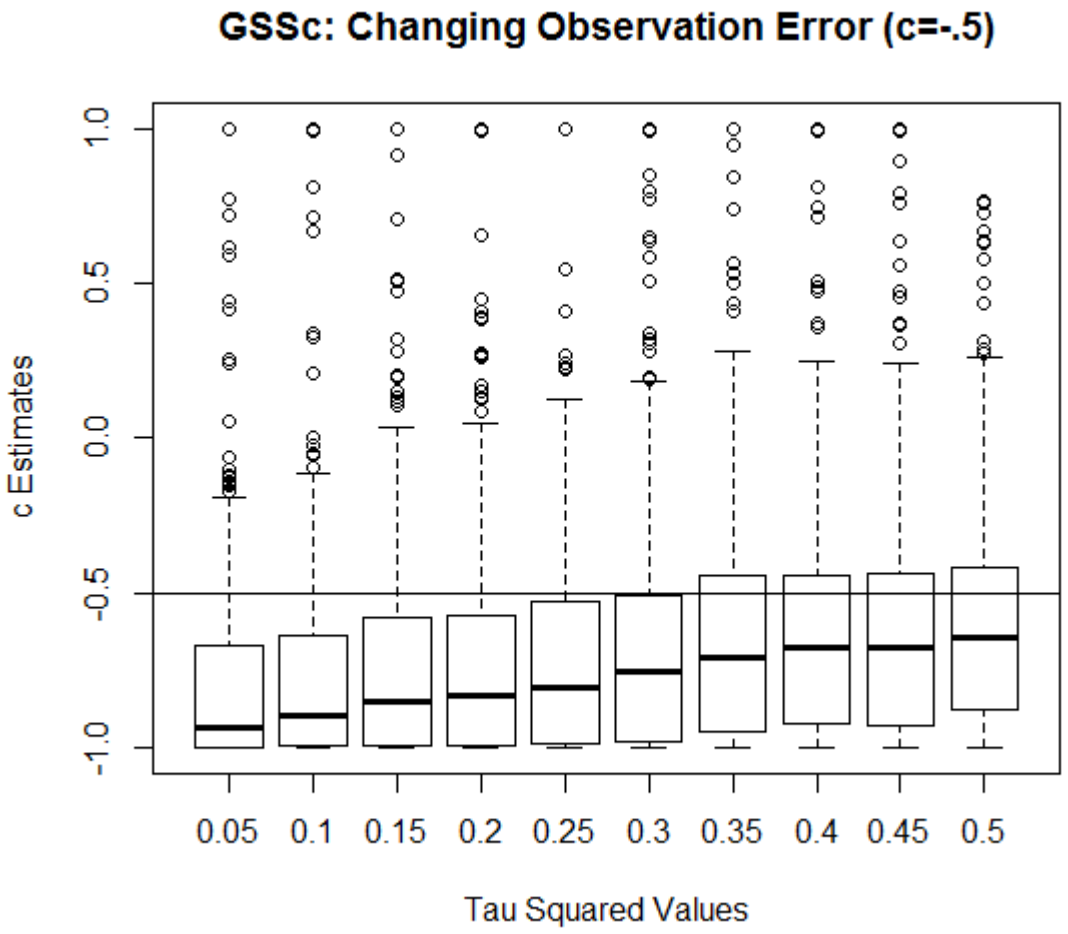


Fig B.2.7. As observation error is added to the GSSc when $c = -.5$ in 0.05 increments from 0.05 to 0.5, there are extreme estimates of c throughout, along with, interestingly, decreasing bias and increasing variance.

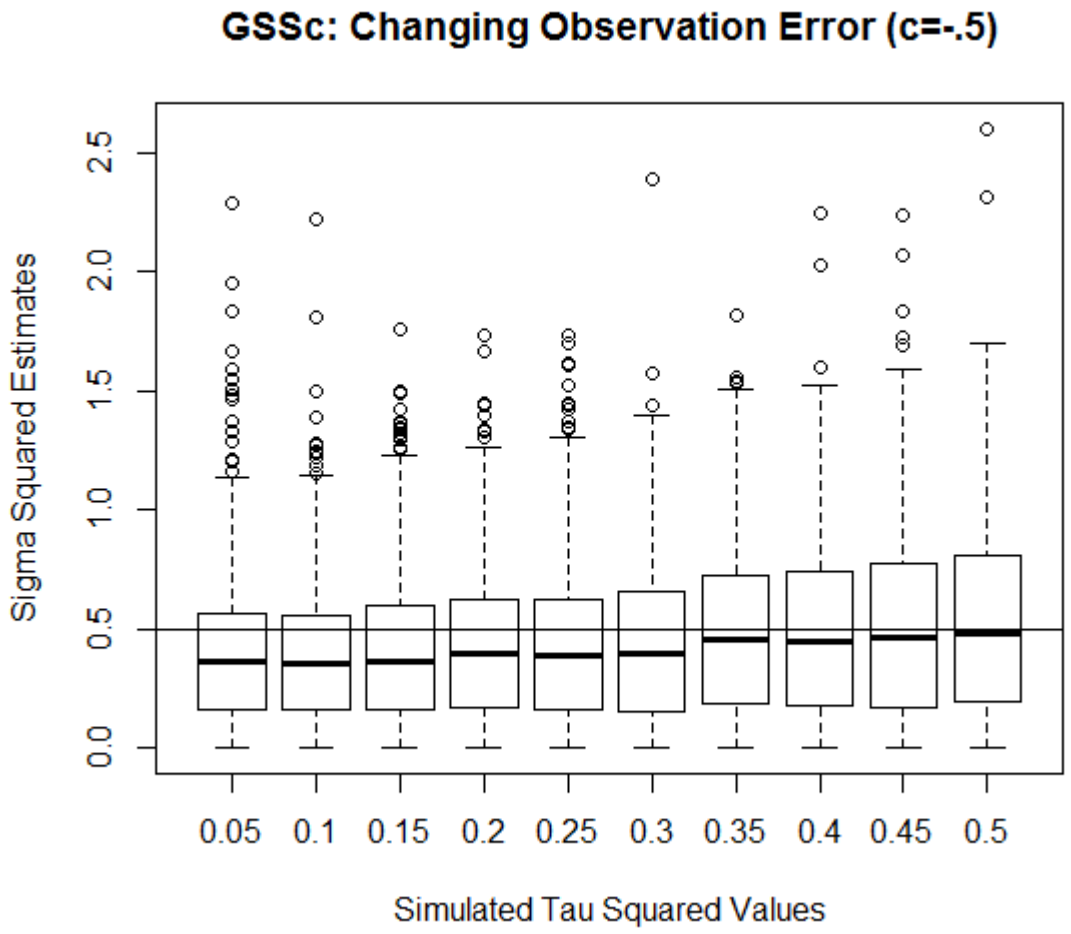


Fig B.2.8. As observation error is added to the GSSc when $c = -.5$ in 0.05 increments from 0.05 to 0.5, there are extreme estimates of σ^2 throughout, ranging all the way up to nearly 3, along with slightly decreasing bias and increasing variance.

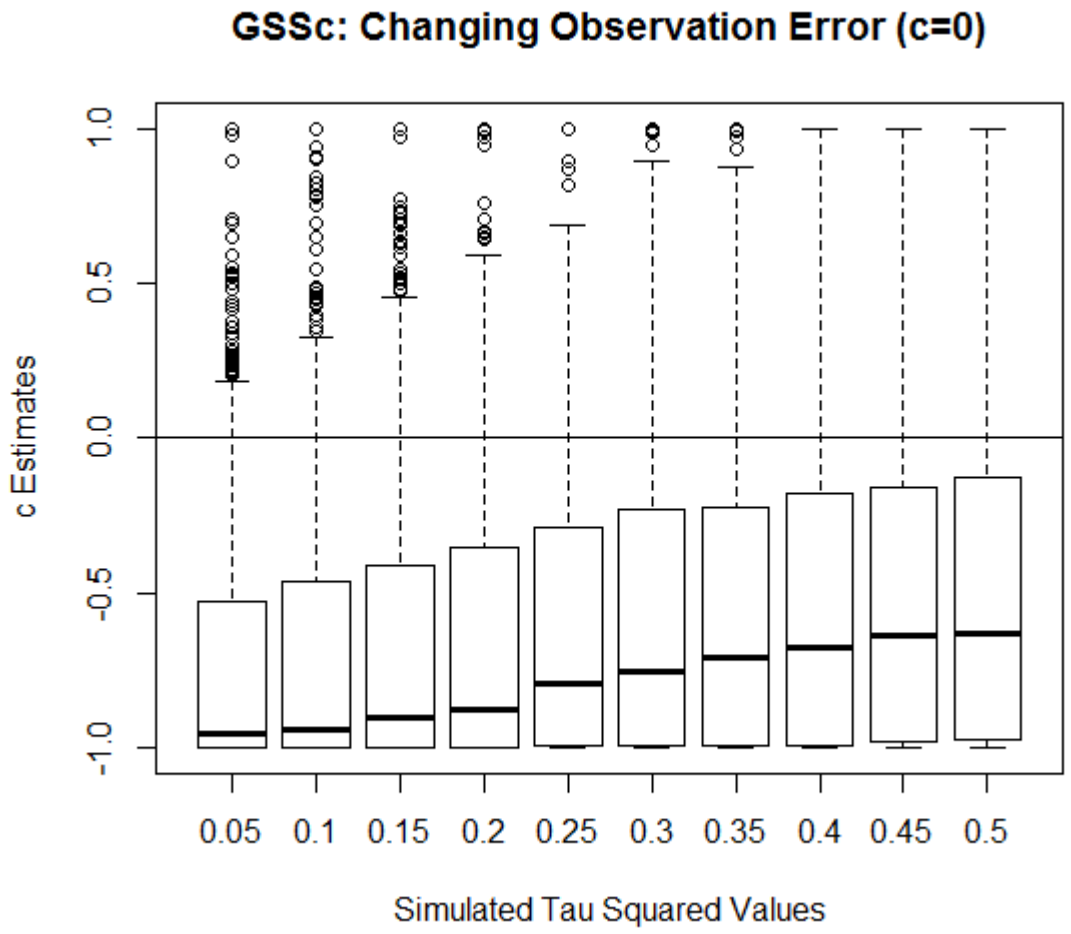


Fig B.2.9. As observation error is added to the GSSc when $c = 0$ in 0.05 increments from 0.05 to 0.5, there are extreme estimates of c throughout, along with decreasing though very pronounced bias and increasing variance. Estimates range over the entire parameter space of c as observation error increases.

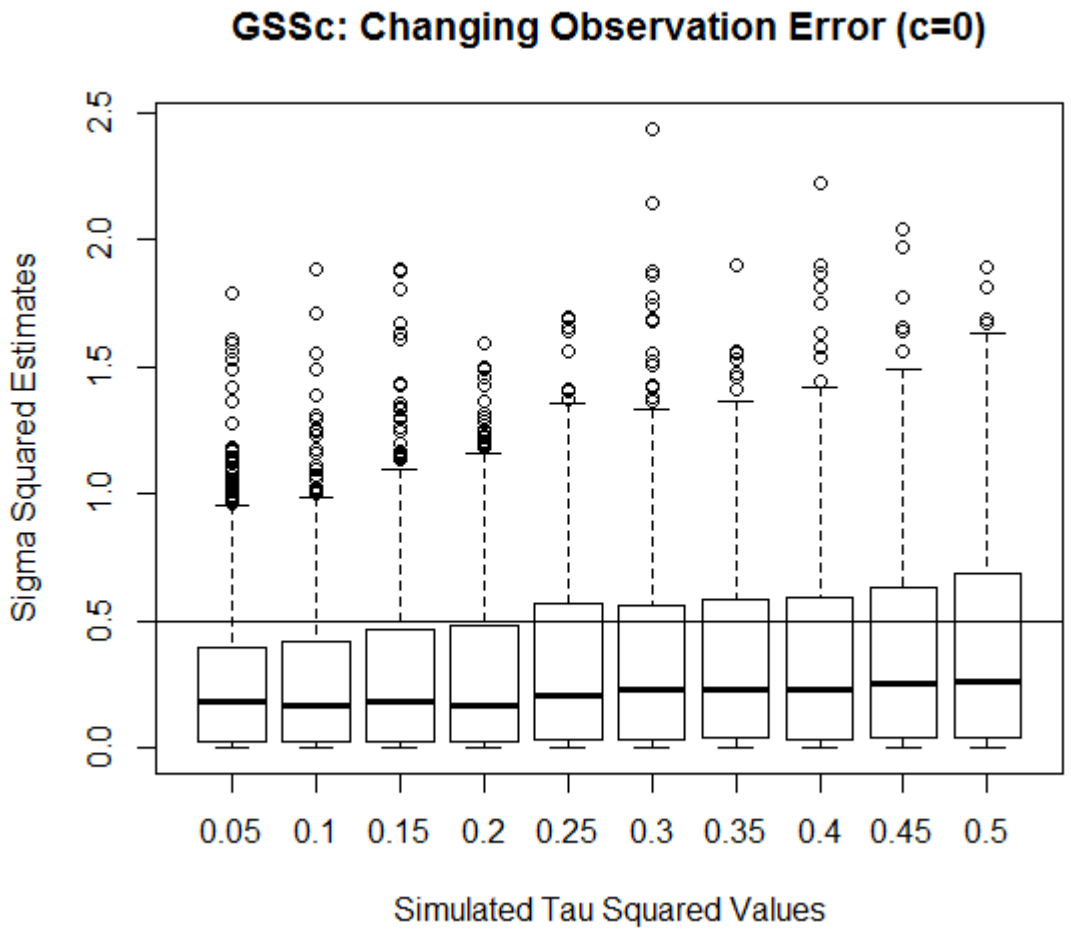


Fig B.2.10. As observation error is added to the GSSc when $c = 0$ in 0.05 increments from 0.05 to 0.5, there are extreme estimates of σ^2 throughout, ranging all the way up to nearly 2.5, along with slightly decreasing, though pronounced, bias and increasing variance.

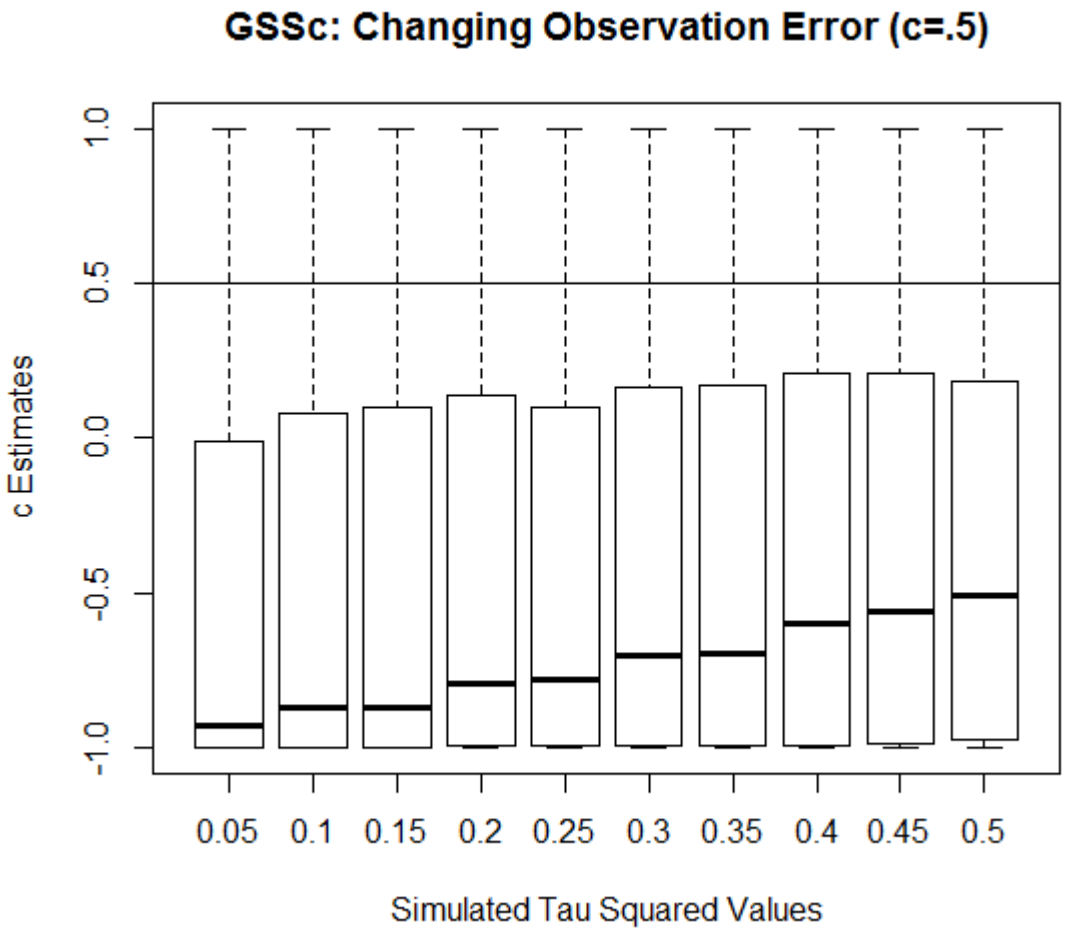


Fig B.2.11. As observation error is added to the GSSc when $c = .5$ in 0.05 increments from 0.05 to 0.5, there are extreme estimates of c throughout, along with decreasing though very pronounced bias. Estimates appear to be converging at nearly the reverse of the true value (-.5 rather than the true value of .5). Estimates range over the entire parameter space of c throughout.

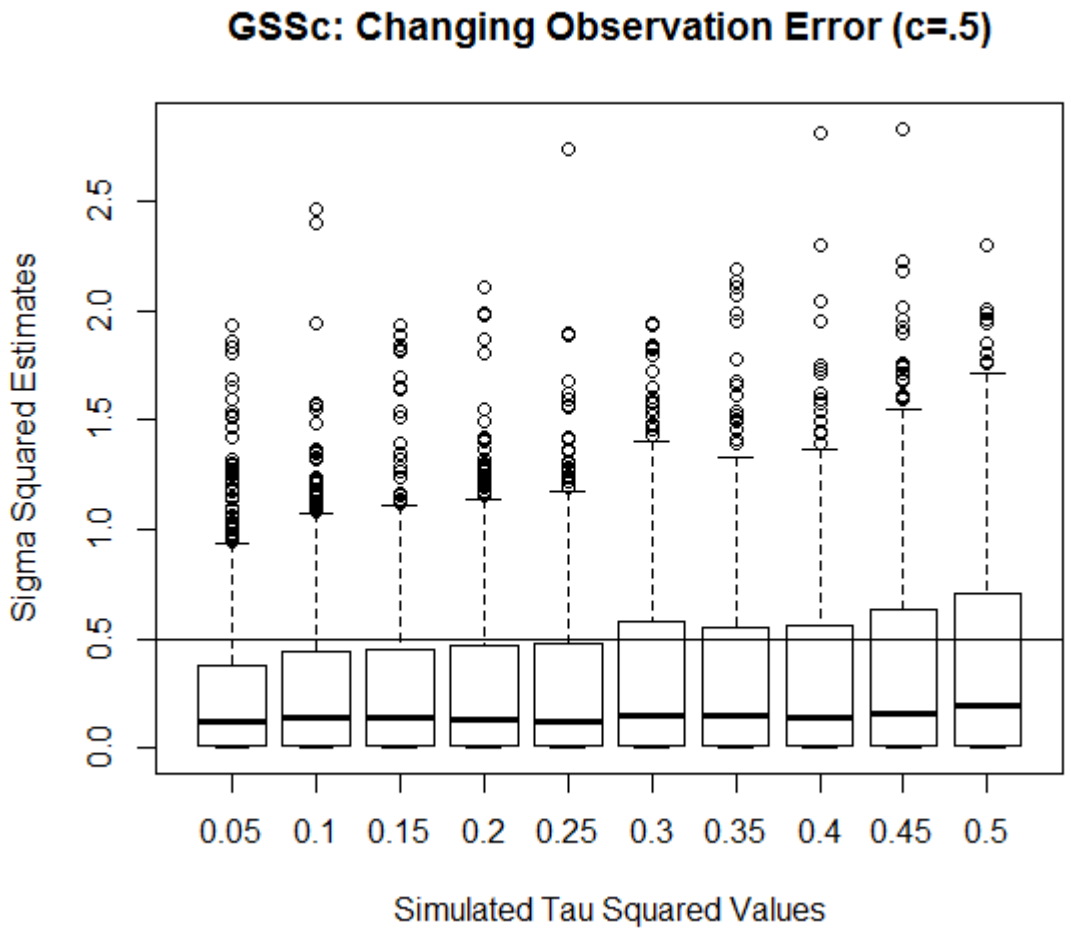


Fig B.2.12. As observation error is added to the GSSc when $c = .5$ in 0.05 increments from 0.05 to 0.5, there are extreme estimates of σ^2 throughout, ranging all the way up to nearly 3, along with pronounced bias and increasing variance.

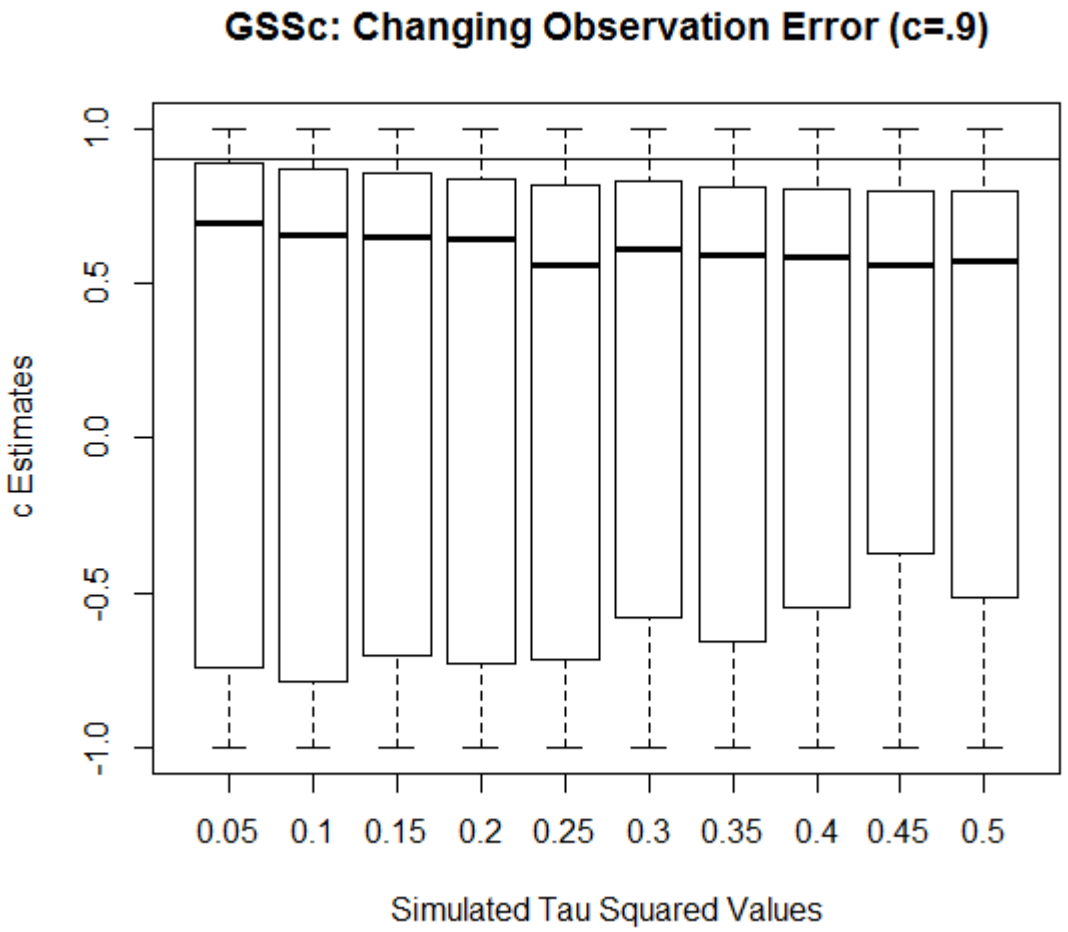


Fig B.2.13. As observation error is added to the GSSc when $c = .9$ in 0.05 increments from 0.05 to 0.5, there are extreme estimates of c throughout, though interestingly the bias is much less pronounced than in other values of c. Estimates range over the entire parameter space of c throughout to such an extreme degree that any confidence interval built using these estimates would likely include the entire parameter space of c.

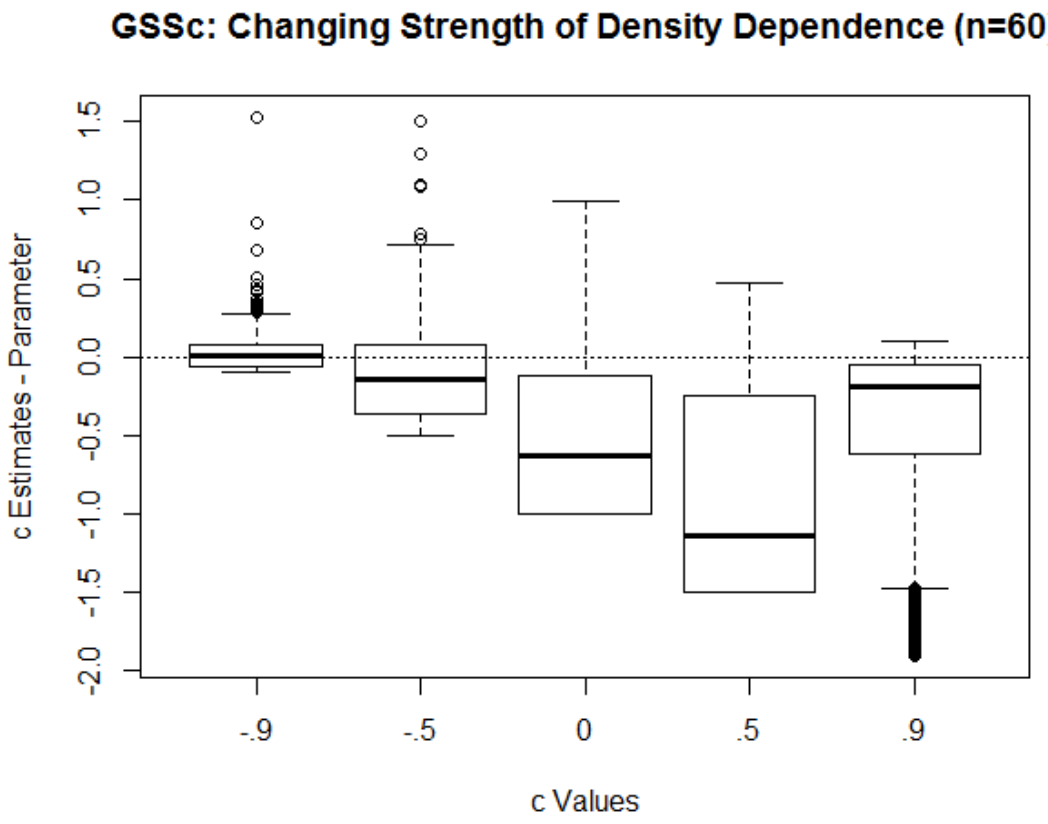


Fig B.2.14. Changing the values of c from $-0.9 to 0.9$ using the GSSc with a time series length of 60 does not appear to help estimability problems. There is still a great deal of bias, especially when $c > 0$, and very large amounts of variability.

Appendix C: Mule Deer Data Example

We have included an example of using the GSSc with a real dataset to show how to recognize potential estimability problems in the data. Complete code, including that for creating profile likelihood plots, is shown below:

Appendix C.1. GSSc with Mule Deer Data Code

```
##### How To Check for Estimability Problems Example: #####
##### GSSc with Mule Deer Data (Peek et al. 2002) #####

# Program to calculate the ML estimates of the GSS model proposed
# by Dennis et al in the paper "Estimating Denisty Dependence,
# Process Noise, and Observation Error" (Ecological Monographs,
# 2006), with covariates and find profile likelihood plots.

# f(y[t]|y[0],y[1],...,y[t-1])=(v[t]^2*2*pi)^(-1/2)*
# exp(-(y[t]-m[t])^2/2*v[t]^2)

# where mt=a+c(m(t-1)+((v(t-1)^2-tao^2)/(v(t-1)^2))*(y(t-1)-m(t-1))
# and vt=c^2*((v(t-1)^2-tao^2)/v(t-1)^2)*tao^2+sigma^2+tao^2

# For stationary distribution:
# m0=a/(1-c)
# v0^2=(sigma^2/(1-c^2))+tao^2
```

```
# For nonstationary distribution:
# m0=x0
# v0^2=tao^2

# Using individual log-likehoods and summing, each
# log f(y[t]|y[t-1],y[t-2],...y[0]=
# -.5*log(2*pi)-.5*log(v[t]^2]-(y[t]-m[t])^2/(2*v[t]^2)

# With covariates:
# f(y[t]|y[0],y[1],...,y[t-1]) is the same as above, but
# m[t]=exp(beta0+beta1*W1+beta2*W2+...+betap*Wp)+c*(m[t-1]
#         +((v[t-1]^2-tao^2)/v[t-1]^2)*(y[t-1]-m[t-1])
# v[t]=c^2*((v[t-1]^2-tao^2)/v[t-1]^2)*tao^2+sigma^2+tao^2
# where p is the number of covariates

# c must be between -1 and 1
# ssq must be > 0
# tsq must be > 0

# data from Peek et al. (XXX)

year=1964:1989
n=c(87.9,86.5,76.1,65.1,47.6,51.4,30.2,34.2,33.6,17.1,14.2,16.4,
    28.1,28.7,33.3,24.8,34.3,23.5,21.8,22.2,28.2,23.2,17.1,15)
canopy=c(27,29,31,33,35,37,39,41,43,45,46,48.5,51,52.5,56,58.5,61,
    63.5,66,68.5,69,71.5,73,74.5,76)
forage=c(11.43,10.42,9.55,8.81,8.18,7.64,7.18,6.78,6.44,6.14,6.01,
```

```
    5,5.73,5.37,5.14,5,4.89,4.38,4.73,4.67,4.65,4.6,4.58,4.56,4.54)
april=c(-0.964,1.482,-0.554,1.161,-1.107,-0.607,-0.893,-0.517,
    -0.304,-1.125,0.232,-0.25,-1.214,-1.25,1.642,1.071,0.032,0.411,
    -0.25,1.161,0.554,-0.893,-1.125,0.339)
june=c(0.833,0.417,0.645,-0.667,1.542,-1.083,0.146,-0.354,-1.479,
    -1.375,0.5,-0.917,1.813,-0.208,-0.979,0.125,-1.125,1.708,-0.75,
    0.042,-0.083,-0.333,0.625,-1.063)
august=c(2.581,1.068,-0.77,1.527,-0.77,-0.77,-0.541,-0.649,-0.716,
    -0.432,-0.189,2.905,-0.554,-0.081,0.621,-0.77,-0.77,0.068,0.203,
    1.324,-0.676,-0.73,-0.77,-0.189)
october=c(-1.206,-0.945,-0.137,-0.123,1.548,1,-0.247,0.808,1.219,
    -0.548,1.055,-1.082,-0.712,-1.26,2.753,-0.589,0.452,0.562,
    -0.206,1.384,-0.082,-1.192,-1.26,0.233)
december=c(-0.492,0.411,-0.579,-0.315,1.234,-0.406,-0.31,-0.228,
    -0.036,-0.193,-0.614,-1.036,0.569,4.858,-0.492,-0.234,1.832,
    0.452,1.802,-0.487,-0.772,0.893,-0.523,-0.929)

q=length(n)-1
k=length(n)

likelihood=function(theta,y){
    y=log(n)
    beta0=(theta[1])
    beta1=(theta[2])
    beta2=(theta[3])
    beta3=(theta[4])
    beta4=(theta[5])
```

```
    beta5=(theta[6])
    beta6=(theta[7])
    beta7=(theta[8])
    cc=2*exp(-exp(theta[9]))-1
    ssq=exp(theta[10])
    tsq=exp(theta[11])
    vv=vector(mode="numeric",k)
    m=vector(mode="numeric",k)
    lpdf=vector(mode="numeric",k)
    m[1]=exp(beta0+beta1*canopy[1]+beta2*forage[1]+beta3*april[1]+
        beta4*june[1]+beta5*august[1]+beta6*october[1]+
        beta7*december[1])/(1-cc)
    vv[1]=(ssq/(1-cc^2))+tsq
    lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
    for(t in 1:q){
       m[t+1]=exp(beta0+beta1*canopy[t+1]+beta2*forage[t+1]+
           beta3*april[t+1]+beta4*june[t+1]+beta5*august[t+1]+
           beta6*october[t+1]+beta7*december[t+1])+cc*(m[t]+
           ((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
       vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
       lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
           (2*vv[t+1]))
     }
    loglike=-sum(lpdf)
    return(loglike)
}
```

```
# To enter starting values into optim, need to convert to theta
# values
# For the purpose of practice problem, using calculated values from
# example in 2006 paper (a=0.3929, c=0.7934, ssq=0.09726,
# tsq=0.2315)
beta0_0=0
beta1_0=-0.1
beta2_0=-0.1
beta3_0=0
beta4_0=0.1
beta5_0=0.2
beta6_0=0
beta7_0=0.1
c0=0.8
ssq0=0.1
tsq0=0.2

# Optimize for ML estimates
ml=optim(par=c(beta0_0,beta1_0,beta2_0,beta3_0,beta4_0,beta5_0,
    beta6_0,beta7_0,log(-log((c0+1)/2)),log(ssq0),log(tsq0)),
    likelihood,NULL,method="Nelder-Mead")
results=c(ml$par[1],ml$par[2],ml$par[3],ml$par[4],ml$par[5],
    ml$par[6],ml$par[7],ml$par[8], 2*exp(-exp(ml$par[9]))-1,
    exp(ml$par[10]),exp(ml$par[11]),-ml$val)

beta0.ml=results[1]
beta1.ml=results[2]
```

```
beta2.ml=results[3]
beta3.ml=results[4]
beta4.ml=results[5]
beta5.ml=results[6]
beta6.ml=results[7]
beta7.ml=results[8]
c.ml=results[9]
sigmasq.ml=results[10]
taosq.ml=results[11]
loglike.ml=results[12]


# User sets parameter intervals for profile plots and total
# number of horizontal axis increments here.
beta0lo=-0.2; # Low value of "beta0"
beta0hi=0.2; # High value of "beta0", etc.
beta1lo=-0.2; beta1hi=0.2;beta2lo=-0.2; beta2hi=0.2;
beta3lo=-0.2; beta3hi=0.2;beta4lo=-0.2; beta4hi=0.2;
beta5lo=-0.2; beta5hi=0.2;beta6lo=-0.2; beta6hi=0.2;
beta7lo=-0.2; beta7hi=0.2; cclo=0.1;cchi=.98;
ssqlo=0.01;ssqhi=0.4;tsqlo=0.1;tsqhi=0.3;
nincs=100; # Number of increments for the profile plots.

library(MASS); # loads miscellaneous functions (ginv, etc.).

# Sets parameter values for profile likelihoods.
beta0vals=seq(beta0lo,beta0hi,by=((beta0hi-beta0lo)/nincs))
```

```
beta1vals=seq(beta1lo,beta1hi,by=((beta1hi-beta1lo)/nincs))
beta2vals=seq(beta2lo,beta2hi,by=((beta2hi-beta2lo)/nincs))
beta3vals=seq(beta3lo,beta3hi,by=((beta3hi-beta3lo)/nincs))
beta4vals=seq(beta4lo,beta4hi,by=((beta4hi-beta4lo)/nincs))
beta5vals=seq(beta5lo,beta5hi,by=((beta5hi-beta5lo)/nincs))
beta6vals=seq(beta6lo,beta6hi,by=((beta6hi-beta6lo)/nincs))
beta7vals=seq(beta7lo,beta7hi,by=((beta7hi-beta7lo)/nincs))
ccvals=seq(cclo,cchi,by=((cchi-cclo)/nincs))
ssqvals=seq(ssqlo,ssqhi,by=((ssqhi-ssqlo)/nincs))
tsqvals=seq(tsqlo,tsqhi,by=((tsqhi-tsqlo)/nincs))

# These vectors will eventually hold the profiles.
profile.beta0=beta0vals
profile.beta1=beta1vals
profile.beta2=beta2vals
profile.beta3=beta3vals
profile.beta4=beta4vals
profile.beta5=beta5vals
profile.beta6=beta6vals
profile.beta7=beta7vals
profile.cc=ccvals
profile.ssq=ssqvals
profile.tsq=tsqvals

# Log-likelihoods for profiles: first the parameter "a" is
# fixed, then "c" is fixed, and so on.
```

```
# "beta0" is a vector of fixed values.
negloglike.beta0.ml=function(theta,parval,y){
    y=log(n)
    beta0=parval
    beta1=(theta[1])
    beta2=(theta[2])
    beta3=(theta[3])
    beta4=(theta[4])
    beta5=(theta[5])
    beta6=(theta[6])
    beta7=(theta[7])
    cc=2*exp(-exp(theta[8]))-1
    ssq=exp(theta[9])
    tsq=exp(theta[10])
    vv=vector(mode="numeric",k)
    m=vector(mode="numeric",k)
    lpdf=vector(mode="numeric",k)
    m[1]=exp(beta0+beta1*canopy[1]+beta2*forage[1]+beta3*april[1]+
        beta4*june[1]+beta5*august[1]+beta6*october[1]+
        beta7*december[1])/(1-cc)
    vv[1]=(ssq/(1-cc^2))+tsq
    lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
    for(t in 1:q){
        m[t+1]=exp(beta0+beta1*canopy[t+1]+beta2*forage[t+1]+
            beta3*april[t+1]+beta4*june[t+1]+beta5*august[t+1]+
            beta6*october[t+1]+beta7*december[t+1])+cc*(m[t]+
            ((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
```

```
        vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
        lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
            (2*vv[t+1]))
      }
    loglike=-sum(lpdf)
    return(loglike)
}

# "beta1" is a vector of fixed values.
negloglike.beta1.ml=function(theta,parval,y){
    y=log(n)
    beta0=(theta[1])
    beta1=parval
    beta2=(theta[2])
    beta3=(theta[3])
    beta4=(theta[4])
    beta5=(theta[5])
    beta6=(theta[6])
    beta7=(theta[7])
    cc=2*exp(-exp(theta[8]))-1
    ssq=exp(theta[9])
    tsq=exp(theta[10])
    vv=vector(mode="numeric",k)
    m=vector(mode="numeric",k)
    lpdf=vector(mode="numeric",k)
    m[1]=exp(beta0+beta1*canopy[1]+beta2*forage[1]+beta3*april[1]+
        beta4*june[1]+beta5*august[1]+beta6*october[1]+
```

```
        beta7*december[1])/(1-cc)
    vv[1]=(ssq/(1-cc^2))+tsq
    lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
    for(t in 1:q){
       m[t+1]=exp(beta0+beta1*canopy[t+1]+beta2*forage[t+1]+
           beta3*april[t+1]+beta4*june[t+1]+beta5*august[t+1]+
           beta6*october[t+1]+beta7*december[t+1])+cc*(m[t]+
           ((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
       vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
       lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
           (2*vv[t+1]))
     }
    loglike=-sum(lpdf)
    return(loglike)
}
# "beta2" is a vector of fixed values.
negloglike.beta2.ml=function(theta,parval,y){
    y=log(n)
    beta0=(theta[1])
    beta1=(theta[2])
    beta2=parval
    beta3=(theta[3])
    beta4=(theta[4])
    beta5=(theta[5])
    beta6=(theta[6])
    beta7=(theta[7])
    cc=2*exp(-exp(theta[8]))-1
```

```
    ssq=exp(theta[9])
    tsq=exp(theta[10])
    vv=vector(mode="numeric",k)
    m=vector(mode="numeric",k)
    lpdf=vector(mode="numeric",k)
    m[1]=exp(beta0+beta1*canopy[1]+beta2*forage[1]+beta3*april[1]+
        beta4*june[1]+beta5*august[1]+beta6*october[1]+
        beta7*december[1])/(1-cc)
    vv[1]=(ssq/(1-cc^2))+tsq
    lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
    for(t in 1:q){
        m[t+1]=exp(beta0+beta1*canopy[t+1]+beta2*forage[t+1]+
            beta3*april[t+1]+beta4*june[t+1]+beta5*august[t+1]+
            beta6*october[t+1]+beta7*december[t+1])+cc*(m[t]+
            ((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
        vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
        lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
            (2*vv[t+1]))
     }
    loglike=-sum(lpdf)
    return(loglike)
}
# "beta3" is a vector of fixed values.
negloglike.beta3.ml=function(theta,parval,y){
    y=log(n)
    beta0=(theta[1])
    beta1=(theta[2])
```

```
beta2=(theta[3])
beta3=parval
beta4=(theta[4])
beta5=(theta[5])
beta6=(theta[6])
beta7=(theta[7])
cc=2*exp(-exp(theta[8]))-1
ssq=exp(theta[9])
tsq=exp(theta[10])
vv=vector(mode="numeric",k)
m=vector(mode="numeric",k)
lpdf=vector(mode="numeric",k)
m[1]=exp(beta0+beta1*canopy[1]+beta2*forage[1]+beta3*april[1]+
    beta4*june[1]+beta5*august[1]+beta6*october[1]+
    beta7*december[1])/(1-cc)
vv[1]=(ssq/(1-cc^2))+tsq
lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
for(t in 1:q){
   m[t+1]=exp(beta0+beta1*canopy[t+1]+beta2*forage[t+1]+
       beta3*april[t+1]+beta4*june[t+1]+beta5*august[t+1]+
       beta6*october[t+1]+beta7*december[t+1])+cc*(m[t]+
       ((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
    vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
    lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
       (2*vv[t+1]))
 }
loglike=-sum(lpdf)
```

```
    return(loglike)
}
# "beta4" is a vector of fixed values.
negloglike.beta4.ml=function(theta,parval,y){
    y=log(n)
    beta0=(theta[1])
    beta1=(theta[2])
    beta2=(theta[3])
    beta3=(theta[4])
    beta4=parval
    beta5=(theta[5])
    beta6=(theta[6])
    beta7=(theta[7])
    cc=2*exp(-exp(theta[8]))-1
    ssq=exp(theta[9])
    tsq=exp(theta[10])
    vv=vector(mode="numeric",k)
    m=vector(mode="numeric",k)
    lpdf=vector(mode="numeric",k)
    m[1]=exp(beta0+beta1*canopy[1]+beta2*forage[1]+beta3*april[1]+
        beta4*june[1]+beta5*august[1]+beta6*october[1]+
        beta7*december[1])/(1-cc)
    vv[1]=(ssq/(1-cc^2))+tsq
    lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
    for(t in 1:q){
        m[t+1]=exp(beta0+beta1*canopy[t+1]+beta2*forage[t+1]+
            beta3*april[t+1]+beta4*june[t+1]+beta5*august[t+1]+
```

```
            beta6*october[t+1]+beta7*december[t+1])+cc*(m[t]+
            ((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
        vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
        lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
            (2*vv[t+1]))
    }
    loglike=-sum(lpdf)
    return(loglike)
}
# "beta5" is a vector of fixed values.
negloglike.beta5.ml=function(theta,parval,y){
    y=log(n)
    beta0=(theta[1])
    beta1=(theta[2])
    beta2=(theta[3])
    beta3=(theta[4])
    beta4=(theta[5])
    beta5=parval
    beta6=(theta[6])
    beta7=(theta[7])
    cc=2*exp(-exp(theta[8]))-1
    ssq=exp(theta[9])
    tsq=exp(theta[10])
    vv=vector(mode="numeric",k)
    m=vector(mode="numeric",k)
    lpdf=vector(mode="numeric",k)
    m[1]=exp(beta0+beta1*canopy[1]+beta2*forage[1]+beta3*april[1]+
```

```
        beta4*june[1]+beta5*august[1]+beta6*october[1]+
        beta7*december[1])/(1-cc)
    vv[1]=(ssq/(1-cc^2))+tsq
    lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
    for(t in 1:q){
       m[t+1]=exp(beta0+beta1*canopy[t+1]+beta2*forage[t+1]+
           beta3*april[t+1]+beta4*june[t+1]+beta5*august[t+1]+
           beta6*october[t+1]+beta7*december[t+1])+cc*(m[t]+
           ((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
       vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
       lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
           (2*vv[t+1]))
     }
    loglike=-sum(lpdf)
    return(loglike)
}
# "beta6" is a vector of fixed values.
negloglike.beta6.ml=function(theta,parval,y){
    y=log(n)
    beta0=(theta[1])
    beta1=(theta[2])
    beta2=(theta[3])
    beta3=(theta[4])
    beta4=(theta[5])
    beta5=(theta[6])
    beta6=parval
    beta7=(theta[7])
```

```
    cc=2*exp(-exp(theta[8]))-1
    ssq=exp(theta[9])
    tsq=exp(theta[10])
    vv=vector(mode="numeric",k)
    m=vector(mode="numeric",k)
    lpdf=vector(mode="numeric",k)
    m[1]=exp(beta0+beta1*canopy[1]+beta2*forage[1]+beta3*april[1]+
        beta4*june[1]+beta5*august[1]+beta6*october[1]+
        beta7*december[1])/(1-cc)
    vv[1]=(ssq/(1-cc^2))+tsq
    lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
    for(t in 1:q){
        m[t+1]=exp(beta0+beta1*canopy[t+1]+beta2*forage[t+1]+
            beta3*april[t+1]+beta4*june[t+1]+beta5*august[t+1]+
            beta6*october[t+1]+beta7*december[t+1])+cc*(m[t]+
            ((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
        vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
        lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
            (2*vv[t+1]))
     }
    loglike=-sum(lpdf)
    return(loglike)
}
# "beta7" is a vector of fixed values.
negloglike.beta7.ml=function(theta,parval,y){
    y=log(n)
    beta0=(theta[1])
```

```
beta1=(theta[2])
beta2=(theta[3])
beta3=(theta[4])
beta4=(theta[5])
beta5=(theta[6])
beta6=(theta[7])
beta7=parval
cc=2*exp(-exp(theta[8]))-1
ssq=exp(theta[9])
tsq=exp(theta[10])
vv=vector(mode="numeric",k)
m=vector(mode="numeric",k)
lpdf=vector(mode="numeric",k)
m[1]=exp(beta0+beta1*canopy[1]+beta2*forage[1]+beta3*april[1]+
    beta4*june[1]+beta5*august[1]+beta6*october[1]+
    beta7*december[1])/(1-cc)
vv[1]=(ssq/(1-cc^2))+tsq
lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
for(t in 1:q){
   m[t+1]=exp(beta0+beta1*canopy[t+1]+beta2*forage[t+1]+
       beta3*april[t+1]+beta4*june[t+1]+beta5*august[t+1]+
       beta6*october[t+1]+beta7*december[t+1])+cc*(m[t]+
       ((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
    vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
    lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
       (2*vv[t+1]))
 }
```

```
    loglike=-sum(lpdf)
    return(loglike)
}
# "c" is a vector of fixed values
negloglike.c.ml=function(theta,parval,y){
    y=log(n)
    beta0=(theta[1])
    beta1=(theta[2])
    beta2=(theta[3])
    beta3=(theta[4])
    beta4=(theta[5])
    beta5=(theta[6])
    beta6=(theta[7])
    beta7=(theta[8])
    cc=parval
    ssq=exp(theta[9])
    tsq=exp(theta[10])
    vv=vector(mode="numeric",k)
    m=vector(mode="numeric",k)
    lpdf=vector(mode="numeric",k)
    m[1]=exp(beta0+beta1*canopy[1]+beta2*forage[1]+beta3*april[1]+
        beta4*june[1]+beta5*august[1]+beta6*october[1]+
        beta7*december[1])/(1-cc)
    vv[1]=(ssq/(1-cc^2))+tsq
    lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
    for(t in 1:q){
        m[t+1]=exp(beta0+beta1*canopy[t+1]+beta2*forage[t+1]+
```

```
            beta3*april[t+1]+beta4*june[t+1]+beta5*august[t+1]+
            beta6*october[t+1]+beta7*december[t+1])+cc*(m[t]+
            ((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
        vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
        lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
            (2*vv[t+1]))
      }
    loglike=-sum(lpdf)
    return(loglike)
}
# "sigma-squared" is a vector of fixed values.
negloglike.ssq.ml=function(theta,parval,y){
    y=log(n)
    beta0=(theta[1])
    beta1=(theta[2])
    beta2=(theta[3])
    beta3=(theta[4])
    beta4=(theta[5])
    beta5=(theta[6])
    beta6=(theta[7])
    beta7=(theta[8])
    cc=2*exp(-exp(theta[9]))-1
    ssq=parval
    tsq=exp(theta[10])
    vv=vector(mode="numeric",k)
    m=vector(mode="numeric",k)
    lpdf=vector(mode="numeric",k)
```

```
    m[1]=exp(beta0+beta1*canopy[1]+beta2*forage[1]+beta3*april[1]+
        beta4*june[1]+beta5*august[1]+beta6*october[1]+
        beta7*december[1])/(1-cc)
    vv[1]=(ssq/(1-cc^2))+tsq
    lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
    for(t in 1:q){
       m[t+1]=exp(beta0+beta1*canopy[t+1]+beta2*forage[t+1]+
           beta3*april[t+1]+beta4*june[t+1]+beta5*august[t+1]+
           beta6*october[t+1]+beta7*december[t+1])+cc*(m[t]+
           ((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
       vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
       lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
           (2*vv[t+1]))
     }
    loglike=-sum(lpdf)
    return(loglike)
}
# "tau-squared" is a vector of fixed values
negloglike.tsq.ml=function(theta,parval,y){
    y=log(n)
    beta0=(theta[1])
    beta1=(theta[2])
    beta2=(theta[3])
    beta3=(theta[4])
    beta4=(theta[5])
    beta5=(theta[6])
    beta6=(theta[7])
```

```
    beta7=(theta[8])
    cc=2*exp(-exp(theta[9]))-1
    ssq=exp(theta[10])
    tsq=parval
    vv=vector(mode="numeric",k)
   m=vector(mode="numeric",k)
    lpdf=vector(mode="numeric",k)
   m[1]=exp(beta0+beta1*canopy[1]+beta2*forage[1]+beta3*april[1]+
        beta4*june[1]+beta5*august[1]+beta6*october[1]+
        beta7*december[1])/
        (1-cc)
    vv[1]=(ssq/(1-cc^2))+tsq
    lpdf[1]=-.5*log(2*pi)-.5*log(vv[1])-((y[1]-m[1])^2/(2*vv[1]))
    for(t in 1:q){
       m[t+1]=exp(beta0+beta1*canopy[t+1]+beta2*forage[t+1]+
            beta3*april[t+1]+beta4*june[t+1]+beta5*august[t+1]+
            beta6*october[t+1]+beta7*december[t+1])+cc*(m[t]+
            ((vv[t]-tsq)/vv[t])*(y[t]-m[t]))
       vv[t+1]=cc^2*((vv[t]-tsq)/vv[t])*tsq+ssq+tsq
       lpdf[t+1]=-.5*log(2*pi)-.5*log(vv[t+1])-((y[t+1]-m[t+1])^2/
            (2*vv[t+1]))
     }
    loglike=-sum(lpdf)
    return(loglike)
}

for (ii in 1:(nincs+1))
```

```
{
# Calculate profile for "beta0".
GSSbeta0=optim(par=c(beta1_0,beta2_0,beta3_0,beta4_0,beta5_0,
    beta6_0,beta7_0,log(-log((c0+1)/2)),log(ssq0),log(tsq0)),
    negloglike.beta0.ml,NULL,method="Nelder-Mead",
    parval=beta0vals[ii])
profile.beta0[ii]=-GSSbeta0$value

# Calculate profile for "beta1".
GSSbeta1=optim(par=c(beta0_0,beta2_0,beta3_0,beta4_0,beta5_0,
    beta6_0,beta7_0,log(-log((c0+1)/2)),log(ssq0),log(tsq0)),
    negloglike.beta1.ml,NULL,method="Nelder-Mead",
    parval=beta1vals[ii])
profile.beta1[ii]=-GSSbeta1$value

# Calculate profile for "beta2".
GSSbeta2=optim(par=c(beta0_0,beta1_0,beta3_0,beta4_0,beta5_0,
    beta6_0,beta7_0,log(-log((c0+1)/2)),log(ssq0),log(tsq0)),
    negloglike.beta2.ml,NULL,method="Nelder-Mead",
    parval=beta2vals[ii])
profile.beta2[ii]=-GSSbeta2$value

# Calculate profile for "beta3".
GSSbeta3=optim(par=c(beta0_0,beta1_0,beta2_0,beta4_0,beta5_0,
    beta6_0,beta7_0,log(-log((c0+1)/2)),log(ssq0),log(tsq0)),
    negloglike.beta3.ml,NULL,method="Nelder-Mead",
    parval=beta3vals[ii])
```

```
profile.beta3[ii]=-GSSbeta3$value

# Calculate profile for "beta4".
GSSbeta4=optim(par=c(beta0_0,beta1_0,beta2_0,beta3_0,beta5_0,
    beta6_0,beta7_0,log(-log((c0+1)/2)),log(ssq0),log(tsq0)),
    negloglike.beta4.ml,NULL,method="Nelder-Mead",
    parval=beta4vals[ii])
profile.beta4[ii]=-GSSbeta4$value

# Calculate profile for "beta5".
GSSbeta5=optim(par=c(beta0_0,beta1_0,beta2_0,beta3_0,beta4_0,
    beta6_0,beta7_0,log(-log((c0+1)/2)),log(ssq0),log(tsq0)),
    negloglike.beta5.ml,NULL,method="Nelder-Mead",
    parval=beta5vals[ii])
profile.beta5[ii]=-GSSbeta5$value

# Calculate profile for "beta6".
GSSbeta6=optim(par=c(beta0_0,beta1_0,beta2_0,beta3_0,beta4_0,
    beta5_0,beta7_0,log(-log((c0+1)/2)),log(ssq0),log(tsq0)),
    negloglike.beta6.ml,NULL,method="Nelder-Mead",
    parval=beta6vals[ii])
profile.beta6[ii]=-GSSbeta6$value

# Calculate profile for "beta7".
GSSbeta7=optim(par=c(beta0_0,beta1_0,beta2_0,beta3_0,beta4_0,
    beta5_0,beta6_0,log(-log((c0+1)/2)),log(ssq0),log(tsq0)),
    negloglike.beta7.ml,NULL,method="Nelder-Mead",
```

```
    parval=beta7vals[ii])
profile.beta7[ii]=-GSSbeta7$value

# Calculate profile for "c".
GSScc=optim(par=c(beta0_0,beta1_0,beta2_0,beta3_0,beta4_0,beta5_0,
    beta6_0,beta7_0,log(ssq0),log(tsq0)),negloglike.c.ml,NULL,
    method="Nelder-Mead",parval=ccvals[ii])
profile.cc[ii]=-GSScc$value

# Calculate profile for "ssq".
GSSssq=optim(par=c(beta0_0,beta1_0,beta2_0,beta3_0,beta4_0,beta5_0,
    beta6_0,beta7_0,log(-log((c0+1)/2)),log(tsq0)),
    negloglike.ssq.ml,NULL,method="Nelder-Mead",parval=ssqvals[ii])
profile.ssq[ii]=-GSSssq$value

# Calculate profile for "tsq".
GSStsq=optim(par=c(beta0_0,beta1_0,beta2_0,beta3_0,beta4_0,beta5_0,
    beta6_0,beta7_0,log(-log((c0+1)/2)),log(ssq0)),
    negloglike.tsq.ml,NULL,method="Nelder-Mead",parval=tsqvals[ii])
profile.tsq[ii]=-GSStsq$value
}

# Sets highest profile value at zero.
profile.beta0=profile.beta0-max(profile.beta0)
profile.beta1=profile.beta1-max(profile.beta1)
profile.beta2=profile.beta2-max(profile.beta2)
profile.beta3=profile.beta3-max(profile.beta3)
```

```
profile.beta4=profile.beta4-max(profile.beta4)
profile.beta5=profile.beta5-max(profile.beta5)
profile.beta6=profile.beta6-max(profile.beta6)
profile.beta7=profile.beta7-max(profile.beta7)
profile.cc=profile.cc-max(profile.cc)
profile.ssq=profile.ssq-max(profile.ssq)
profile.tsq=profile.tsq-max(profile.tsq)

# Profiles plotted here.
par(cex.lab=1.5, cex.axis=1.5, lwd=2)
layout(matrix(1:6, 3, 2))
plot(beta0vals, profile.beta0, type="l", lty=1, ylab="log like",
   xlab=expression(beta0))
plot(beta2vals, profile.beta2, type="l", lty=1, ylab="log like",
   xlab=expression(beta2))
plot(beta4vals, profile.beta4, type="l", lty=1, ylab="log like",
   xlab=expression(beta4))
plot(beta1vals, profile.beta1, type="l", lty=1, ylab="log like",
   xlab=expression(beta1))
plot(beta3vals, profile.beta3, type="l", lty=1, ylab="log like",
   xlab=expression(beta3))
plot(beta5vals, profile.beta5, type="l", lty=1, ylab="log like",
   xlab=expression(beta5))

par(cex.lab=1.5, cex.axis=1.5, lwd=2)
layout(matrix(1:6, 3, 2))
plot(beta6vals, profile.beta6, type="l", lty=1, ylab="log like",
```

```
    xlab=expression(beta6))
plot(ccvals, profile.cc, type="l", lty=1, ylab="loglike",
    xlab=expression(c))
plot(tsqvals, profile.tsq, type="l", lty=1, ylab="log like",
    xlab=expression(tau^2))
plot(beta7vals, profile.beta7, type="l", lty=1, ylab="log like",
    xlab=expression(beta7))
plot(ssqvals, profile.ssq, type="l", lty=1, ylab="log like",
    xlab=expression(sigma^2))

# Print results
beta0.ml
beta1.ml
beta2.ml
beta3.ml
beta4.ml
beta5.ml
beta6.ml
beta7.ml
c.ml
sigmasq.ml
taosq.ml
loglike.ml
```

Appendix C.2. Profile Likelihood Plots Examination

When we look at the profile likelihood plots created using the code in Appendix C.1, it is clear that something unexpected is occuring (Figure C.2.1 and C.2.2). Plots do not show

smooth unimodal curves as anticipated. Estimability problems are recognizable in profile likelihood plots by either ridge shaped curves or jagged, multimodal curves, as seen below.

After determining that estimability is a problem with the 7 covariates included in the mule deer data, we examined the data with a single covariate (canopy) and recreated the associated profile plots using modified code from Appendix C.1 (Figure C.2.3). Even with a single covariate, profile plots are clearly showing potential estimability problems. At this point, a simulation study such as that conducted in this study should be conducted to determine limits of estimability of the model.

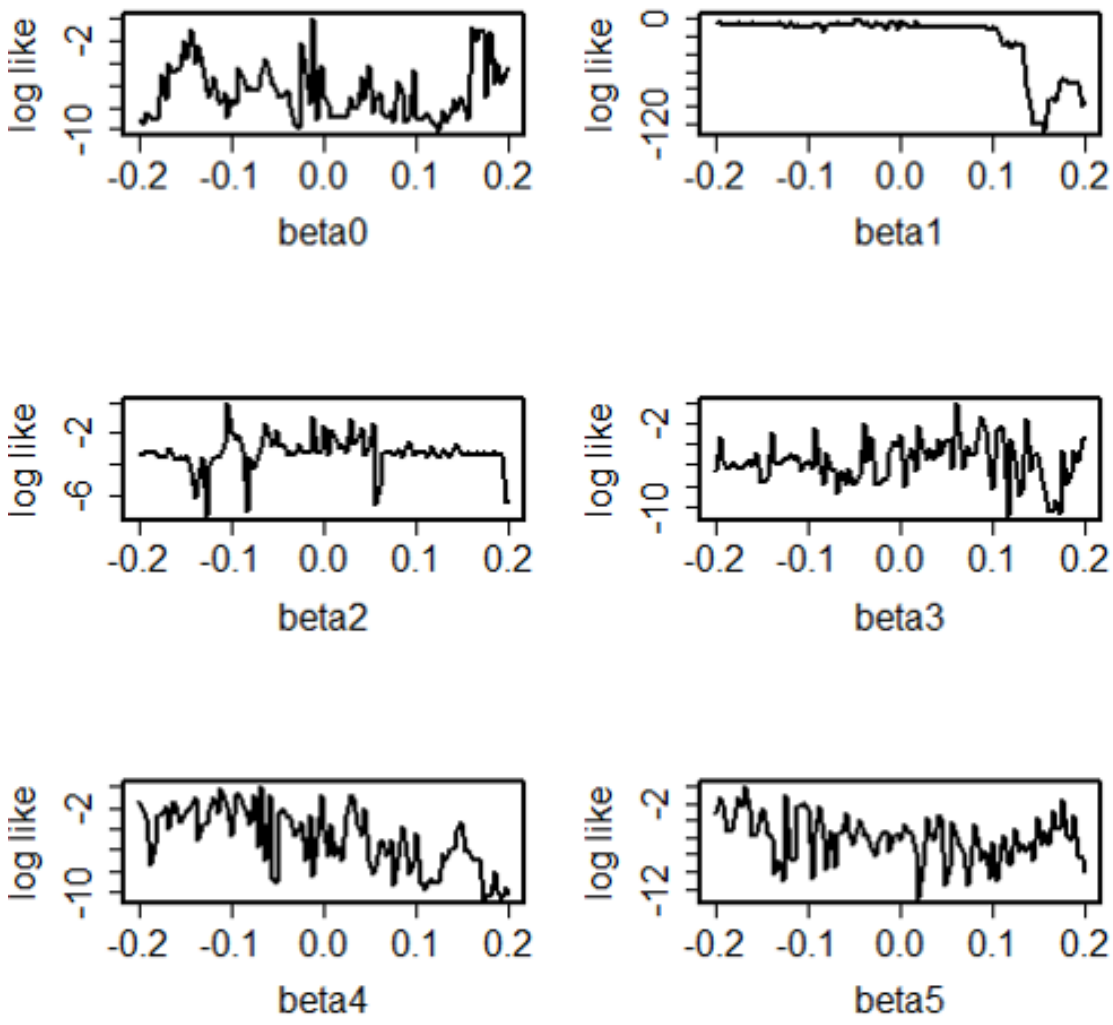


Fig C.2.1. Profile likelihood plots of the first 6 parameters of the GSSc with 7 covariates from the mule deer data show unexpected behavior. Jagged, multimodal curves with no clear maximum value indicate possible estimability problems that should be investigated.

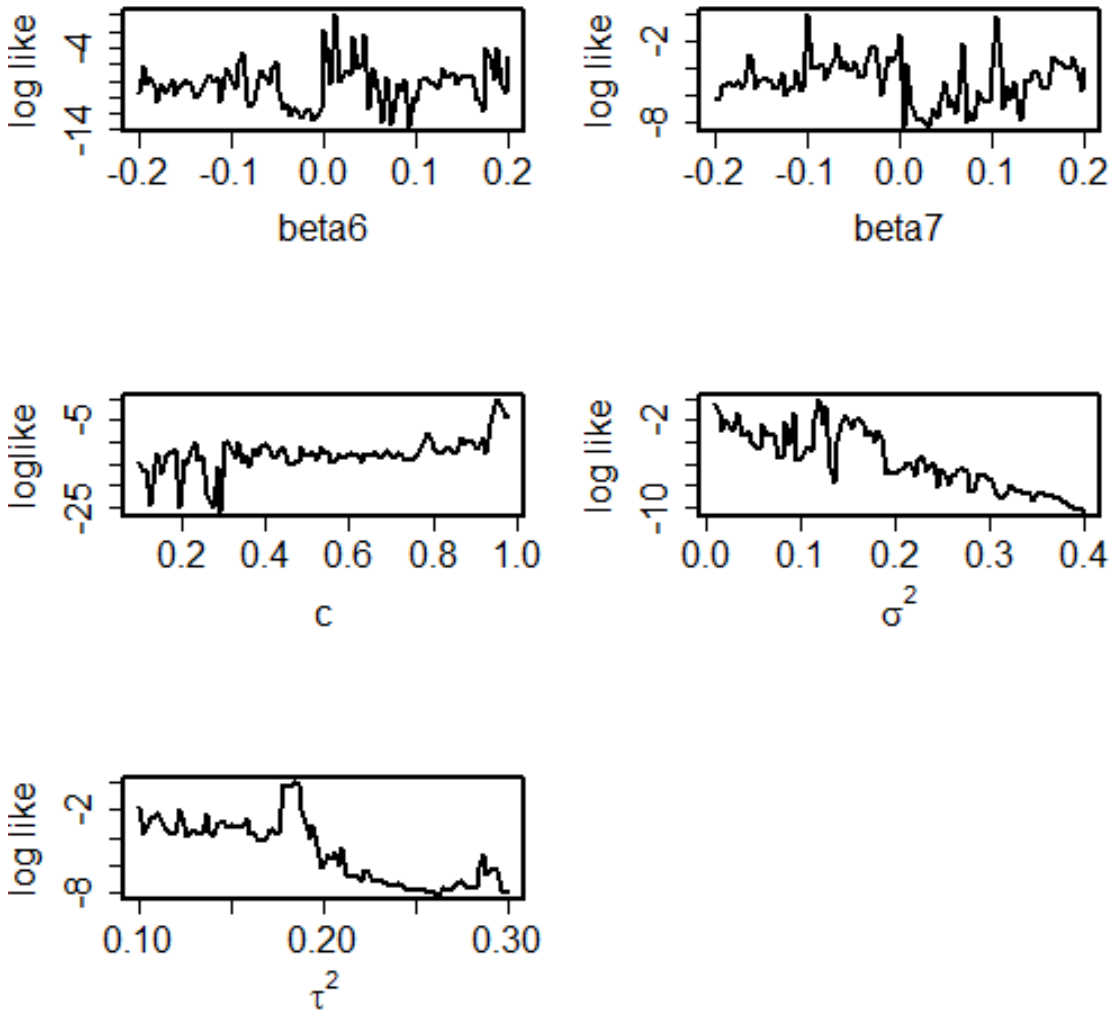


Fig C.2.2. Profile likelihood plots of the last 5 parameters of the GSSc with 7 covariates from the mule deer data show unexpected behavior. Jagged, multimodal curves with no clear maximum value indicate possible estimability problems that should be investigated.

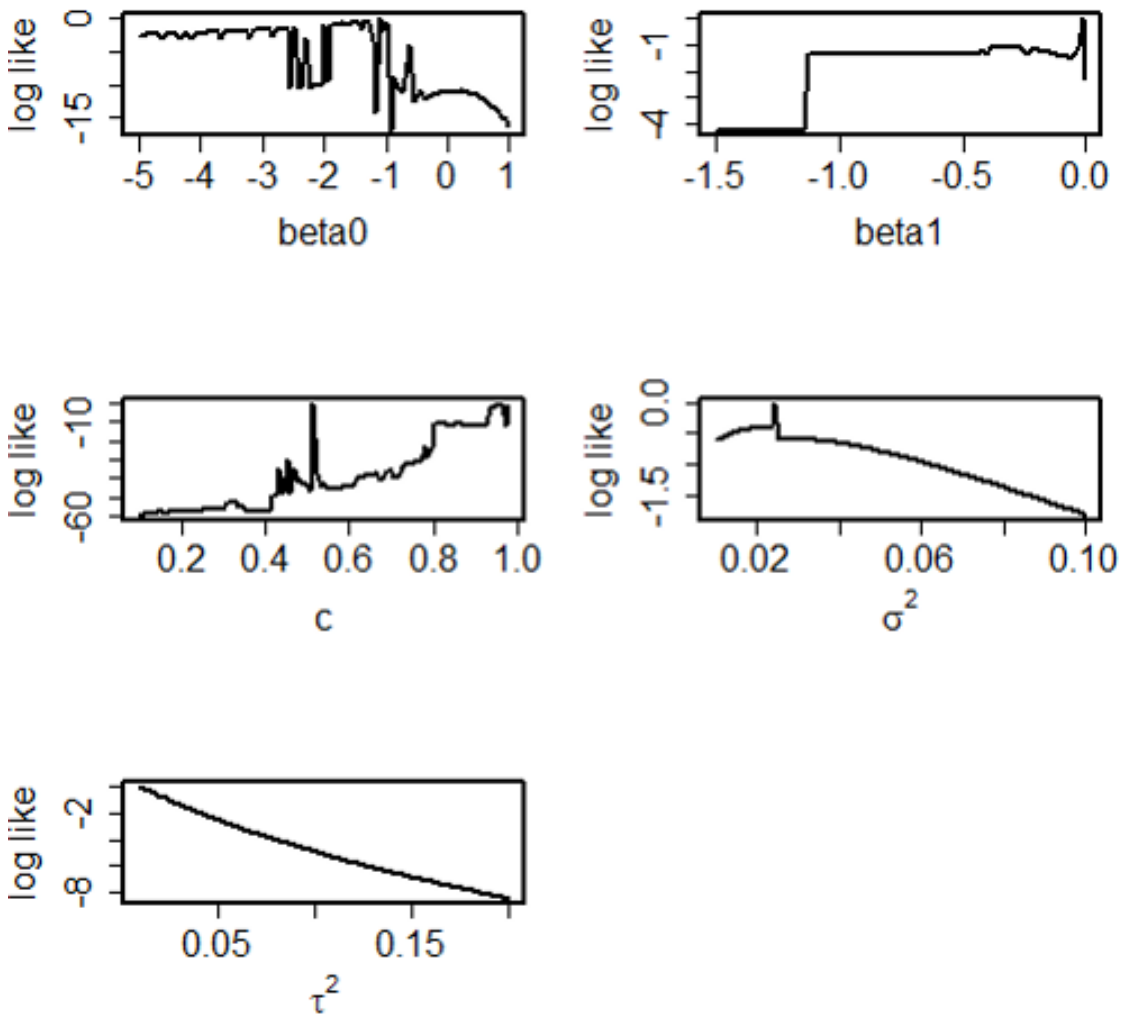


Fig C.2.3. Profile likelihood plots of the GSSc with only the first covariate from the mule deer data continue to show unexpected behavior. Jagged, multimodal curves with no clear maximum value indicate possible estimability problems that should be investigated.