STAT/BIOST 571: Homework 1

Philip Pham

January 25, 2019

Problem 1: Non-parametric linear regression (5 points)

In the context of random $\boldsymbol{X}$ linear regression, consider n independent observations (x_i, Y_i) generated as follows. First, the indepenent variable x_i is selected according to

$$x_i = \begin{cases} 0 & \text{w.p. } 1/3 \\ 1 & \text{w.p. } 1/3 \\ 3 & \text{w.p. } 1/3 \end{cases},$$

then the systematic component of Y_i is set to

$$\mu_i = \begin{cases} 0.6 & \text{if } x_i = 0 \\ 3.6 & \text{if } x_i = 1 \\ 6.8 & \text{if } x_i = 3 \end{cases},$$

and finally $Y_i \sim N(\mu_i, \sigma^2)$ with $\sigma^2 = 1$.

(a) Simulate a single realization from the data-generating mechanism described above (with $n = 20$) and present the data in a scatterplot, with the Y_i on the vertical axis and the x_i on the horizontal axis. Fit a simple linear regression model of the form

$$Y_i = \beta_0 + x_i\beta_1 + \epsilon_i$$

by ordinary least squares and overlay the fitted regression line on your scatterplot. Comment on the approporiateness of fitting a linear model to these data.

Solution: See Figure 1.

The line isn't a very good fit. It overestimates when $x = 0$ and underestimates when $x = 1$. This isn't surprising since the linear model is incorrect.

(b) Let $\hat{\beta}_1$ be the estimated slope parameter from fitting the model above, and define β_1 to be the quantity for which $\hat{\beta}_1$ is consistent, in the limit as $n \to \infty$. Using the result on Slide 1.18 of the course lecture notes, calculate β_1 exactly.

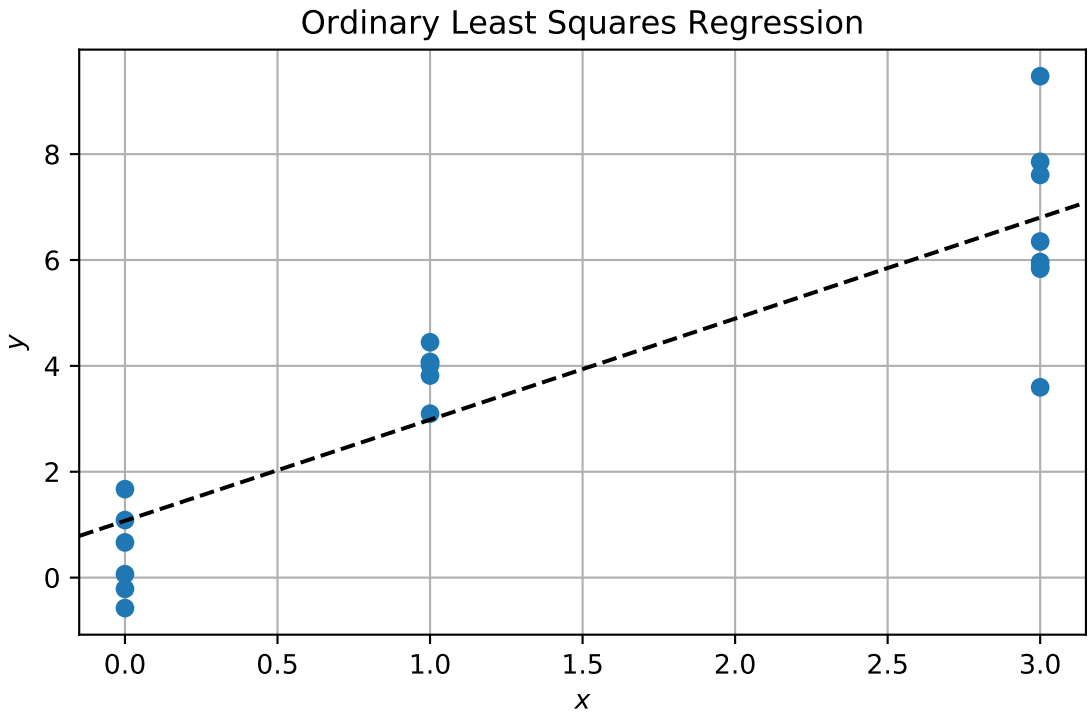


Figure 1: Ordinary least squares for data according to the model in Problem 1.

Solution: $\beta_0 = 1$ and $\boxed{\beta_1 = 2.}$

Let $\mathbb{P}$ be the probability measure over x and y. Let $\phi\left(x \mid \mu, \sigma^2\right)$ be the density of the normal distribution with mean μ and variance σ^2. Consider the integral,

$$\begin{aligned} L(\gamma) &= \int (y - (\gamma_0 + \gamma_1 x))^2 \, \mathrm{d}\,\mathbb{P}(x, y) \\ &= \frac{1}{3}\int_{-\infty}^{\infty} (y - \gamma_0)^2 \, \phi(y \mid 0.6, 1) \, \mathrm{d}\, y + \\ &\quad \frac{1}{3}\int_{-\infty}^{\infty} (y - \gamma_0 - \gamma_1)^2 \, \phi(y \mid 3.6, 1) \, \mathrm{d}\, y + \\ &\quad \frac{1}{3}\int_{-\infty}^{\infty} (y - \gamma_0 - 3\gamma_1)^2 \, \phi(y \mid 6.8, 1) \, \mathrm{d}\, y \\ &= \frac{1}{3}\left(62.56 - 22\gamma_0 - 48\gamma_1 + 3\gamma_0^2 + 10\gamma_1^2 + 8\gamma_0\gamma_1\right). \end{aligned}$$

If we let $\beta = \arg\min_\gamma L(\gamma)$. Taking the derviative, we setting $\frac{\partial L}{\partial \gamma}(\beta) = 0$ to obtain $\beta_0 = 1$ and $\beta_1 = 2$.

(c) Download White's seminal paper from 1980 on sandwich estimation from the course website . Read (at least) the first few pages and explain how the value of β_1 you derived in part (b) also follows from Lemma 1 of the paper.

Solution: In fitting the model, we are assuming that the linear model is correct and Assumption 1 of the paper holds.

Assumption 2 is satisfied so since $\mathbb{E}\left[\epsilon_i^2\right] = 1$ for all i and the expected covariance matrix $= \bar{M}_n = n^{-1}\sum_{i=1}^n \mathbb{E}\left[X_i^2\right]$ is just a matrix 1×1 with a single positive entry, so it is certainly nonsingular is with a positive determinant.

Thus, Lemma 1 tells us that $\hat{\beta} \xrightarrow{\text{a.s.}} \beta$, where $\hat{\beta} = (X^\intercal X)^{-1} X^\intercal Y$.

Since $\hat{\beta}$ is also the maximum likelihood estimate which minimizes mean squared error, it converges almost surely to the value of β that minimizes expected mean squared error.

(d) For finite n, we cannot discuss properties of the expectation of $\hat{\beta}_1$, at least not in the conventional sense. Explain why not.

Solution: Note that $\hat{\beta} = (X^\intercal X)^{-1} X^\intercal Y$.

If the linear model were correct, that is, $Y = X\beta + \epsilon$, taking the expection would give

$$\mathbb{E}\left[\hat{\beta}\right] = \beta + \mathbb{E}\left[(X^\intercal X)^{-1} X^\intercal \epsilon\right] = \beta. \tag{1}$$

However, This calculation is incorrect when Y depends on X in a nonlinear way as with these data.

For the remainder of this problem, condition on realizations of the data for which there exist i and i' such that $x_i \neq x_{i'}$.

(e) Conduct a simulation study to assess the bias of $\hat{\beta}_1$ for estimating β_1. Plot the estimated bias as a function of n (for $n = 2, 4, 6, \ldots, 28, 30, 35, \ldots, 95, 100$), and overlay the standard error of $\hat{\beta}_1$ as a function of n. Note that for this problem you should estimate the standard error of $\hat{\beta}_1$ based on the standard deviation of $\hat{\beta}_1$ across simulated realizations of the data, not using any further exact or asymptotic calculations.

Solution: See Table 1 for the slope estimates $\hat{\beta}_1$ and standard error estimates $\hat{\sigma}$ for different values of n.

These data are plotted in Figure 2.

(f) Comment on your findings in part (e). Is the bias in estimating β_1 surprising? Why or why not? How concerned should one be about this bias?

Solution: The bias is not suprising. As discussed in Part (d), the usual method of proving that the estimate is unbiased does not work since the true model is not linear.

The bias decreases quickly with increasing n, so it is not too concerning with reasonable sample sizes. Moreover, the standard error is larger than the bias, so in the bias-variance

n	$\hat{\beta}_1$	$\hat{\sigma}$	$\mathbb{E}[\hat{\beta}_1]$
2	2.222531	1.115256	2.222222
4	2.119261	0.752701	2.119358
6	2.058301	0.536735	2.057653
8	2.028134	0.408784	2.028077
10	2.014462	0.332531	2.014561
12	2.008274	0.284690	2.008200
14	2.005376	0.253067	2.005023
16	2.003600	0.231102	2.003320
18	2.002070	0.213511	2.002338
20	2.001587	0.200356	2.001732
22	2.001091	0.189500	2.001336
24	2.001169	0.180668	2.001063
26	2.000937	0.172259	2.000867
28	2.000748	0.165419	2.000721
30	2.000182	0.159203	2.000610
30	2.000505	0.159288	2.000610
35	2.000618	0.146462	2.000424
40	2.000300	0.136427	2.000312
45	2.000245	0.127994	2.000239
50	2.000259	0.121297	2.000189
55	2.000184	0.115340	2.000154
60	2.000254	0.110308	2.000127
65	2.000014	0.105664	2.000107
70	2.000246	0.101802	2.000091
75	2.000068	0.098277	2.000079
80	2.000033	0.094885	2.000069
85	2.000091	0.092020	2.000060
90	2.000172	0.089272	2.000054
95	1.999983	0.087027	2.000048
100	1.999924	0.084749	2.000043

Table 1: The estimate for the slope ($\hat{\beta}_1$) and standard error for the estimate ($\hat{\sigma}$) were each calculated with 10^6 trials. The last column $\mathbb{E}\left[\hat{\beta}_1\right]$ was calculated numerically by enumerating over the possible draws of X, which is detailed in Part (g).

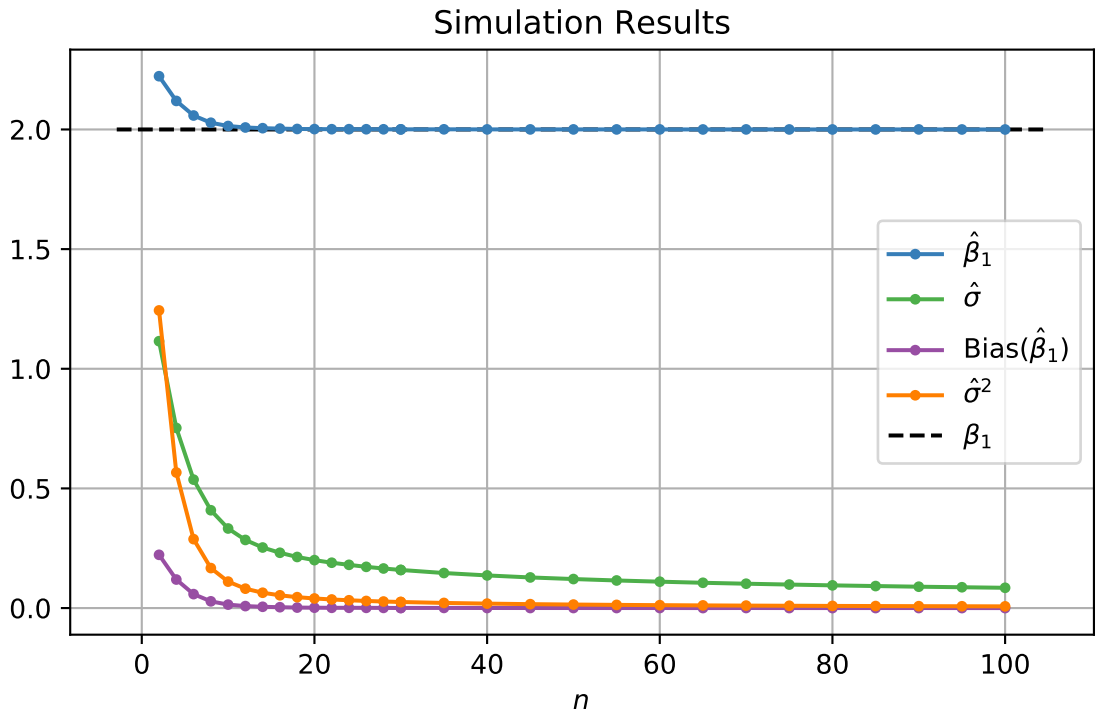


Figure 2: Data from Table 1 along with asymptotic slope (β_1), the bias ($\hat{\beta}_1 - \beta_1$), and variance ($\hat{\sigma}^2$).

decomposition,

$$\begin{aligned}
\mathbb{E}\left[\left(\hat{\beta}_1 - \beta_1\right)^2\right] &= \mathbb{E}\left[\left(\hat{\beta}_1 - \beta_1\right)^2\right] = \mathbb{E}\left[\hat{\beta}_1^2\right] - 2\mathbb{E}\left[\hat{\beta}_1\right]\beta_1 + \beta_1^2 \\
&= \operatorname{var}\left(\hat{\beta}_1\right) + \mathbb{E}\left[\hat{\beta}_1\right]^2 - 2\mathbb{E}\left[\hat{\beta}_1\right]\beta_1 + \beta_1^2 \\
&= \left(\mathbb{E}\left[\hat{\beta}_1\right] - \beta_1\right)^2 + \operatorname{var}\left(\hat{\beta}_1\right) \\
&= \left[\operatorname{Bias}\left(\hat{\beta}_1\right)\right]^2 + \operatorname{var}\left(\hat{\beta}_1\right),
\end{aligned}$$

the variance term which is the standard error squared will dominate. Indeed, according to Figure 2, the variance is much larger than the bias when n is small. When n is large, both the bias and variance are small, so neither is too concerning.

(g) It is possible to derive exact values for the biases estimated in part (d) by enumerating all possible draws of $x_1, \ldots, x_n$. Do this calculation for $n = 2, 4, 6, 8$. Report your numerical findings and overlay them on your plot from part (e). Show how you can do the calculation entirely by hand for $n = 2$ (you may use a computer for matrix algebra). Although it is highly advisable to use a computer for this calculation for $n > 2$, simulation results are not sufficient. Your calculation of the bias must be exactly correct up to the computer's floating point precision.

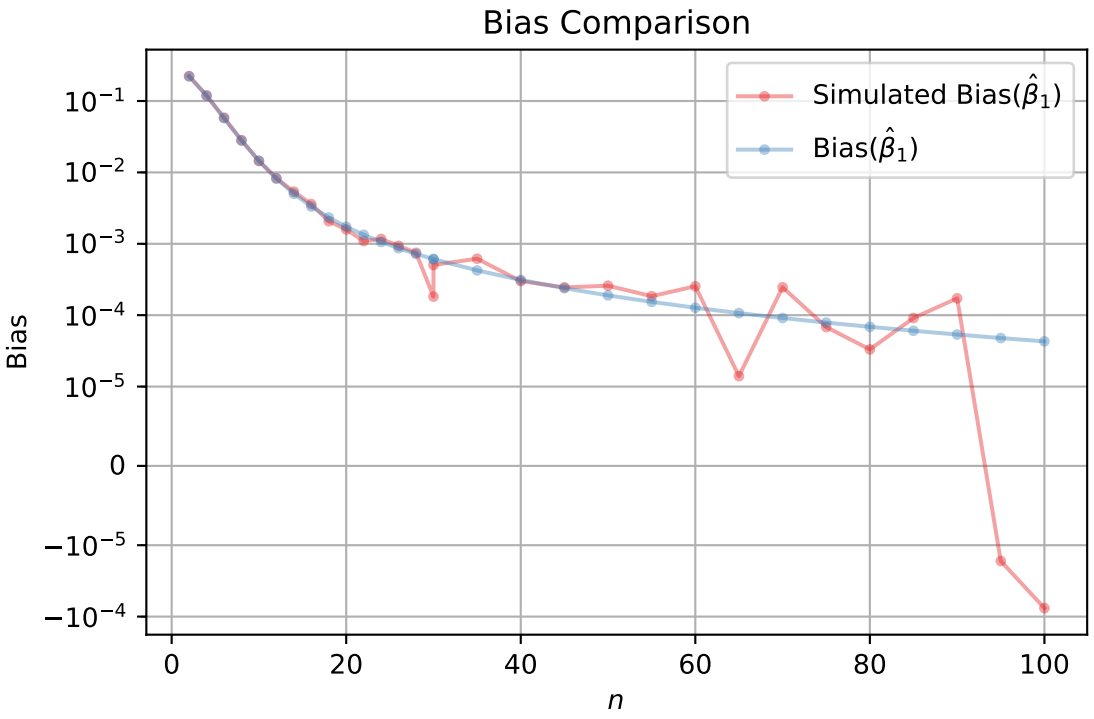


Figure 3: Comparison of the simulated bias from Part (e) against the exact calculation in Part (g).

Solution: $\mathbb{E}\left[\hat{\beta}\right]$ can be efficiently calculated by noticing that

$$\mathbb{E}\left[\hat{\beta} \mid X\right] = \mathbb{E}\left[\left(X^\intercal X\right)^{-1} X^\intercal Y \mid X\right] = \left(X^\intercal X\right)^{-1} X^\intercal \mu. \tag{2}$$

is the same for different permuations of X.

Assume that each $X_{(j_0,j_1,j_3)} \in \mathbb{R}^n$ is a vector of j_0 0s, j_1 1s, and j_3 3s, where $j_0+j_1+j_3 = n$ and $j_0, j_1, j_3 \in \{0, 1, \ldots, n-1\}$. The triplets (j_0, j_1, j_3) can be seen as coming from the multinomial distribution, so

$$\mathbb{P}\left((j_0, j_1, j_3)\right) = \frac{n!}{j_0! j_1! j_3!} \left(\frac{1}{3}\right)^n. \tag{3}$$

Therefore, we have that

$$\begin{aligned}
\mathbb{E}\left[\hat{\beta}\right] &= \mathbb{E}\left[\mathbb{E}\left[\hat{\beta} \mid X\right]\right] \\
&= \sum_{\left\{(j_0,j_1,j_3)\in\{0,1,\ldots,n-1\}^3 \mid j_0+j_1+j_3=n\right\}} \mathbb{P}\left((j_0, j_1, j_3 = n)\right) \left(X^\intercal X\right)^{-1} X^\intercal \mu \\
&= \sum_{\left\{(j_0,j_1,j_3)\in\{0,1,\ldots,n-1\}^3 \mid j_0+j_1+j_3=n\right\}} \frac{n!}{j_0! j_1! j_3!} \left(\frac{1}{3}\right)^n \left(X^\intercal X\right)^{-1} X^\intercal \mu.
\end{aligned}$$

The results of this calculation can be seen in the last column of Table 1. The biases from the simulation are plotted against the exact calculation in Figure 3.

Both the table and figure show that the simulation and calulations agree. There appears to be more noise with larger n, however.

Problem 2: Average slope interpretation of generalized least squares (5 points)

Consider simple linear regression with a single covariate and an intercept, such that we are fitting

$$E(Y_i) = \beta_0 + \beta_1 x_i$$

with scalar observations $Y_1, \ldots, Y_n$. As we saw on slide 1.18, the OLS estimator of β_1 can be written as a weighted average of the pairwise slopes $(Y_i - Y_j)/(x_i - x_j)$. This is called the "average slope" representation.

(a) Suppose that $\mathbf{\Sigma}$, the covariance of $\boldsymbol{Y}$, is a diagonal matrix with known entries $\sigma_1^2, \ldots, \sigma_n^2$. Derive the weights for an "average slope" representation for the optimal generalized least squares (GLS) estimate of β_1 (see exercise 8.1 in the Wakefield text if you are not sure of the form of the optimal GLS estimate). A formal proof is optional, but you should provide a convincing justification for the specific form of your answer. HINT: One way to approach this problem is to think about what happens when the σ_i^2 are all rational numbers, i.e., they can be expressed as fractions.

Solution: The likelihood function is

$$L_{X,y,\Sigma}(\beta) = f_\beta(y) = \frac{1}{\sqrt{\det(2\pi\Sigma)}} \exp\left(-\frac{1}{2}(y - X\beta)^\intercal \Sigma^{-1}(y - X\beta)\right). \tag{4}$$

Maximizing this likelihood by taking the log and derivative, we estimate β to obtain the maximum likelihood estimate $\hat{\beta} = \left(X^\intercal \Sigma^{-1} X\right)^{-1} X^\intercal \Sigma^{-1} y$.
Expanding this out, we find that

$$\hat{\beta} = \begin{pmatrix} \left(\sum_{i=1}^n \frac{x_i^2}{\sigma_i^2}\right)\left(\sum_{i=1}^n \frac{y_i}{\sigma_i^2}\right) - \left(\sum_{i=1}^n \frac{x_i}{\sigma_i^2}\right)\left(\sum_{i=1}^n \frac{x_i y_i}{\sigma_i^2}\right) \\ \left(\sum_{i=1}^n \frac{1}{\sigma_i^2}\right)\left(\sum_{i=1}^n \frac{x_i y_i}{\sigma_i^2}\right) - \left(\sum_{i=1}^n \frac{x_i}{\sigma_i^2}\right)\left(\sum_{i=1}^n \frac{y_i}{\sigma_i^2}\right) \end{pmatrix} \Bigg/ \left(\left(\sum_{i=1}^n \frac{1}{\sigma_i^2}\right)\left(\sum_{i=1}^n \frac{x_i^2}{\sigma_i^2}\right) - \left(\sum_{i=1}^n \frac{x_i}{\sigma_i^2}\right)^2\right).$$

Let $\bar{x} = \sum_{i=1}^n \frac{1}{\sigma_i^2} x_i \Big/ \sum_{i=1}^n \frac{1}{\sigma_i^2}$ denote the weighted mean. We can rewrite our estimate for the slope

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n \frac{1}{\sigma_i^2}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n \frac{1}{\sigma_i^2}(x_i - \bar{x})^2} = \frac{\sum_{i=1}^n \frac{1}{\sigma_i^2} \sum_{j=1}^n \frac{1}{\sigma_j^2} \sum_{k=1}^n \frac{1}{\sigma_k^2}(x_i - x_j)(y_i - y_k)}{\sum_{i=1}^n \frac{1}{\sigma_i^2} \sum_{j=1}^n \frac{1}{\sigma_j^2} \sum_{k=1}^n \frac{1}{\sigma_k^2}(x_i - x_j)(x_i - x_k)}.$$

Now with some algebra,

$$\begin{aligned}\sum_{i=1}^{n}\frac{1}{\sigma_i^2}\sum_{j=1}^{n}\frac{1}{\sigma_j^2}\sum_{k=1}^{n}\frac{1}{\sigma_k^2}\left(x_i-x_j\right)\left(y_i-y_k\right) &= \sum_{j=1}^{n}\frac{1}{\sigma_j^2}\sum_{i=1}^{n}\frac{1}{\sigma_i^2}\sum_{k=1}^{n}\frac{1}{\sigma_k^2}\left(x_i-x_j\right)\left(y_i-y_k\right)\\ &= \left(\sum_{k=1}^{n}\frac{1}{\sigma_k^2}\right)\sum_{i=1}^{n}\frac{1}{\sigma_i^2}\sum_{j=1}^{n}\frac{1}{\sigma_j^2}\left(x_i-x_j\right)y_i\\ &= \frac{1}{2}\left(\sum_{k=1}^{n}\frac{1}{\sigma_k^2}\right)\sum_{i=1}^{n}\frac{1}{\sigma_i^2}\sum_{j=1}^{n}\frac{1}{\sigma_j^2}\left(x_i-x_j\right)\left(y_i-y_j\right),\end{aligned}$$

where the last two steps take advantage of the symmetry of the i and j indices. Using this result, we have can write our slope estimate

$$\begin{aligned}\hat{\beta}_1 &= \frac{\sum_{i=1}^{n}\frac{1}{\sigma_i^2}\sum_{j=1}^{n}\frac{1}{\sigma_j^2}\left(x_i-x_j\right)\left(y_i-y_j\right)}{\sum_{i=1}^{n}\frac{1}{\sigma_i^2}\sum_{j=1}^{n}\frac{1}{\sigma_j^2}\left(x_i-x_j\right)^2}\\ &= \boxed{\sum_{i=1}^{n}\sum_{\substack{j=1\\ x_i\neq x_j}}^{n}\left(\frac{\left(x_i-x_j\right)^2}{\sigma_i^2\sigma_j^2}\right)\left(\frac{y_i-y_j}{x_i-x_j}\right)\Bigg/\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\left(x_i-x_j\right)^2}{\sigma_i^2\sigma_j^2}}.\end{aligned}$$

Thus, the unnormalized weight for the slope between i and j is $\boxed{\frac{\left(x_i-x_j\right)^2}{\sigma_i^2\sigma_j^2}}$. The normalized weights are $\boxed{w_{ij}=\frac{\left(x_i-x_j\right)^2}{\sigma_i^2\sigma_j^2}\Bigg/\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\left(x_i-x_j\right)^2}{\sigma_i^2\sigma_j^2}}$.

Intuitively, this make sense. As σ_i^2 and σ_j^2 increase, the weight decreases since our estimated slope is noisier. On the other hand, as $(x_i-x_j)^2$ increases the weight increases because the slope estimate becomes more robust to noise since the effect size is larger. To see this mathematically, note that $\frac{y_i-y_j}{x_i-x_j}=\beta_1+\frac{\sigma_i\epsilon_i-\sigma_j\epsilon_j}{x_i-x_j}$, where $\epsilon_i\sim\mathcal{N}(0,1)$. The error term increases in magnitude as a function of σ_i^2 and σ_j^2 and decreases in magnitude as a function of $(x_i-x_j)^2$.

(b) Now consider what the weights would be in an "average slope" representation for the optimal GLS estimate of β_1, for a general covariance matrix $\mathbf{\Sigma}$. Give some properties of the weights that you would expect to hold. In answering this question, it might be helpful to think about how the weights would change as you vary individual entries in $\mathbf{\Sigma}$. Also discuss any situations where you would expect the weight for a particular pair of observations to be infinite or zero. Note that you are **not** asked to derive an average slope representation for GLS.

Solution: Let $\mathbf{1}$ be a vector a in $\mathbb{R}^n$ of all 1s. Denote the weighted mean, $\bar{x}=\mathbf{1}^\intercal\Sigma^{-1}x/\mathbf{1}^\intercal\Sigma^{-1}\mathbf{1}$
Denote the centered vector, $\tilde{x}=x-\bar{x}\mathbf{1}$.

Then, we have that

$$\hat{\beta}_1=\frac{\tilde{x}^\intercal\Sigma^{-1}\tilde{y}}{\tilde{x}^\intercal\Sigma^{-1}\tilde{x}}=\sum_{i=1}^{n}\sum_{j=1}^{n}\Sigma_{ij}^{-1}\tilde{x}_i\tilde{y}_j\Bigg/\sum_{i=1}^{n}\sum_{j=1}^{n}\Sigma_{ij}^{-1}\tilde{x}_i\tilde{x}_j. \tag{5}$$

Let $\Delta \in \mathbb{R}^{n \times n}$ be the skew-symmetric matrix where $\Delta_{ij} = x_i - x_j$. Let Σ_i denote the ith row or column of the covariance matrix (the matrix is symmetric). We can rewrite our slope estimate as

$$\hat{\beta}_1 = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ x_i \neq x_j}}^{n} \frac{y_i - y_j}{x_i - x_j} (x_i - x_j) \operatorname{tr}\left(\Delta \Sigma_i^{-1} \left(\Sigma_j^{-1}\right)^\intercal\right) \Bigg/ \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i - x_j) \operatorname{tr}\left(\Delta \Sigma_i^{-1} \left(\Sigma_j^{-1}\right)^\intercal\right),$$

where tr is the trace operator. Since

$$\begin{aligned} -(x_i - x_j) \operatorname{tr}\left(\Delta \Sigma_i^{-1} \left(\Sigma_j^{-1}\right)^\intercal\right) &= (x_i - x_j) \sum_{k=1}^{n} \sum_{l=1}^{n} (x_k - x_l) \Sigma_{ik}^{-1} \Sigma_{jl}^{-1} \\ &= (x_i - x_j) \sum_{k=1}^{n} \sum_{l=k+1}^{n} (x_k - x_l) \det \begin{pmatrix} \Sigma_{ik}^{-1} & \Sigma_{il}^{-1} \\ \Sigma_{jk}^{-1} & \Sigma_{jl}^{-1} \end{pmatrix}, \end{aligned} \tag{6}$$

we can see the pairwise slopes are weighted according to the entries from the precision matrix. Moreover, the situation differs from the independent case in that the weight of each slope depends on all possible pairwise differences $x_j - x_l$, which intuitively makes sense since there may be some correlation with the other observations.

Let $\tilde{\rho}_{ij}$ be the partial correlation coefficient between Y_i and Y_j given the remaining Y_k. The partial correlation measures the correlation between observations controlling for cofounders. Entries of the precision matrix are the scaled partial correlations, $\Sigma_{ij}^{-1} = \tilde{\rho}_{ij} \sqrt{\Sigma_{ii}^{-1} \Sigma_{jj}^{-1}}$.

With this identity, we can rewrite unnormalized weights for the slope estimate

$$-(x_i - x_j) \operatorname{tr}\left(\Delta \Sigma_i^{-1} \left(\Sigma_j^{-1}\right)^\intercal\right) = (x_i - x_j) \Sigma_{ii}^{-1} \Sigma_{jj}^{-1} \sum_{k=1}^{n} \sum_{l=k+1}^{n} (x_k - x_l) \det \begin{pmatrix} \tilde{\rho}_{ik} & \tilde{\rho}_{il} \\ \tilde{\rho}_{jk} & \tilde{\rho}_{jl} \end{pmatrix}, \tag{7}$$

Equation 7 gives us insight into how the covariance matrix Σ affects the weights for the pairwise slope between observations i and j.

As before, large differences $x_i - x_j$ increase the magnitude of the weight, and large variances, manifested as small entries of Σ_{ii}^{-1} and Σ_{jj}^{-1}, decrease the magnitude of the weight. Now, the differences $x_j - x_l$ also affect the weight in a similar manner to $x_i - x_j$. The partial correlation coefficients govern the degree of this effect. If there is no partial correlation, that is, $\tilde{\rho}_{ik} = 0$ and $\tilde{\rho}_{il} = 0$ or $\tilde{\rho}_{jk} = 0$ and $\tilde{\rho}_{jl} = 0$, then the term involving $x_k - x_l$ is zeroed out. Of course, when the observations are independent only the terms with $(k, l) = (i, j)$ or $(k, l) = (j, i)$ are nonzero, and we recover the same weights as the previous part.

Problem 3: Cross-sectional/longitudinal effects, partitioning the exposure, mean model misspecification (10 points)

Consider a made-up computer literacy trial similar to the one discussed in lecture (slides 1.35–1.48). There are n subjects, indexed by $i = 1, \ldots, n$, each one of which is observed at m followup times, indexed by $j = 1, \ldots, m$. Let Y_{ij} be the literacy score of subject i at follow-up time j, at which time the subject's

age is x_{ij}. Suppose the design is fixed, meaning that the x_{ij} are all deterministic and known in advance, and assume the true data-generating mechanism can be written

$$Y_{ij} = f(x_{i1}) + \beta_L(x_{ij} - x_{i1}) + \epsilon_{ij} \tag{8}$$

with i.i.d. $\epsilon_{ij} \sim N(0, \sigma^2)$. The unknown function $f(\cdot)$ represents a potentially nonlinear cohort effect and β_L is the coefficient for the linear longitudinal effect. Suppose we try to estimate β_L using the exposure partioning method proposed on slide 1.44 (see, also, Diggle et al. page 16), based on the linear regression estimating equations with the mean model

$$E(Y_{ij}) = \beta_0 + \beta_C x_{i1} + \beta_L(x_{ij} - x_{i1}). \tag{9}$$

(a) Give a sufficient condition on the data-generating mechanism for Y_{ij} in equation (8) to guarantee unbiased estimation of β_L and valid model-based standard errors.

Solution: Let $\tilde{X}$ be the matrix with 1s in the first columns, x_{i1} (repeated as necessary) in the second column and $x_{ij} - x_{i1}$ in the third column. Let Y be a vector of the corresponding literacy scores. We can estimate β_L with the least squares estimator

$$\hat{\beta} = \begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_C \\ \hat{\beta}_L \end{pmatrix} = \left(\tilde{X}^\intercal \tilde{X}\right)^{-1} \tilde{X}^\intercal Y. \tag{10}$$

If we substitute Equation 8 for Y, $\mathbb{E}\left[\hat{\beta}\right] = \beta$ if f is a linear function, that is, $f(x_{i1}) = \beta_0 x_{i1}$ for some constant β_0.

Given f is linear, the covariance matrix for our estimator is

$$\text{var}\left(\hat{\beta}\right) = \sigma^2 \left(\tilde{X}^\intercal \tilde{X}\right)^{-1}. \tag{11}$$

(b) Give a sufficient condition on the choice of follow-up times in the study design to guarantee unbiased estimation of β_L, even if the condition in part (a) is violated. Thinking about when confounding bias is a problem will be helpful here.

Solution: To control for the cofounding factors, we can use randomization. In particular, if follow-up times are selected independently of the subjects' ages, any systematic bias from incorrectly predicting base literacy rates will be eliminated.

To see this, note that if the covariates are independent $\mathbb{E}\left[\tilde{X}^\intercal \tilde{X}\right]$ is a diagonal matrix (without loss of generality assume $\tilde{X}$ is centered), so the coefficients are estimated independently of each other.

(c) Download the modified made-up literacy data from the course web site (you can find these data with the homework files, not in the general dataset section), and generate one or more exploratory plots. Describe how you can graphically see evidence that neither of the above conditions is satisfied.

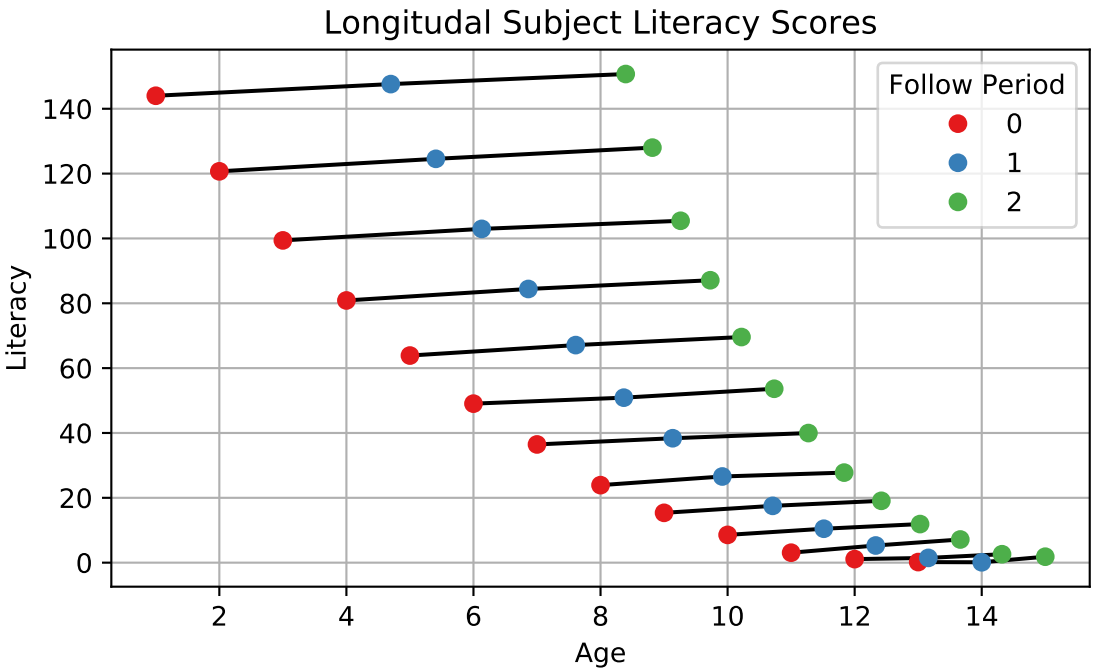


Figure 4: In the computer literacy data, each subject has 3 observations taken at similar follow-up periods.

Solution: See the plotted data in Figure 4.

Looking at the red points, one sees that f is clearly not linear, so the assumption in Part (a) is violated. Each subject was followed up with twice. $x_{i3} - x_{i2}$ appears to be similar to $x_{i2} - x_{i1}$. For older subjects, the follow up periods appear to be shorter. Clearly, the follow up times were not selected independently of the subjects' ages, so the assumption Part (b) is violated.

(d) Propose an alternative regression model that can still be used to unbiasedly estimate β_L and give valid model-based standard errors. Fit this model to the downloaded data using the `lm()` function in `R` and report your findings (point estimate and standard error). Compare to what you obtain by fitting model (9) to the data.

Solution: We can fit a model directly to the difference $Y_{ij} - Y_{i1} = \beta_L (x_{ij} - x_{i1}) + (\epsilon_{ij} - \epsilon_{i1})$, $j \neq 1$. In this way, we discard one-third of our observations. Let $\tilde{X}_3$ be the third column of $\tilde{X}$ and let $\tilde{Y}$ be the vector of $Y_{ij} - Y_{i1}$. Assume that we have discarded the $j = 1$ entries. Then, we can our new model can estimated $\hat{\beta}_L = \left(\tilde{X}_3^\intercal \tilde{X}_3\right)^{-1} \tilde{X}_3^\intercal \tilde{Y}$. Since the expected residual is 0 this estimator is always unbiased.

Fitting this model to the data, we find that $\boxed{\hat{\beta}_L = 1.02185154}$ with corresponding standard error $\boxed{\sqrt{\hat{\text{var}}\left(\hat{\beta}_L\right)} = 0.032418976.}$

If we fit the model in Equation 9, we have $\boxed{\hat{\beta}_L = 1.24721578}$ with corresponding standard

error $\boxed{\sqrt{\hat{\text{var}}\left(\hat{\beta}_L\right)} = 1.06297.}$ Thus, the misspecified model leads to a much higher standard error.

(e) Design and conduct a simulation study to illustrate the following

(a) Model (9) can fail to give unbiased estimates of β_L without at least one of the additional conditions from parts (a) and (b).

(b) Model (9) does give unbiased estimates if either of the additional conditions from parts (a) and (b) is satisfied.

(c) The model you propose in part (d) works, regardless.

You will need to consider a few choice of $f(\cdot)$ and study designs $\{x_{ij}\}$ to illustrate points (i) through (iii). Note that in earlier parts of the problem, you should have argued theoretically that each of these points is true; the purpose of the simulation study is to verify and illustrate your conclusions.

Solution: I fixed $\beta_L = 2$, $\sigma^2 = 1$, and use 64 subjects. If I specified 2 follow-up times for each subject and let $f(x) = 200 * \exp(-x)$, both assumptions are violated. Indeed, fitting the model in Equation 9 led to a biased estimate of 2.16. The alternative model from the previous section handle this case fine and produced an unbiased estimate.

Next, I enforced the condition in Part (a) and made f linear, $f(x) = 200 - 12x$. Both the model in Equation 9 and the alternative model result in unbiased estimates.

When it's enforced that the follow up times are independent of the base age, again both models produce unbiased estimates even if the f is a nonlinear function like $f(x) = 200 * \exp(-x)$.

Code for all these simulations is found in the Appendix.

(f) In your simulation study, what do you notice about the standard error estimates from the model in equation (9) when the condition from part (b) is satisfied but the condition from part (a) is not satisfied? Do sandwich standard errors fix the problem?

Solution: With the usual linear regression estimate for the variance of $\hat{\beta}_L$, the standard errors are overestimated. My simulations show that the 95% confidence interval actually covered the true mean 98.1% of the time.

Sandwich estimation seems to help a little, but the situation is the same with the same: the 95% confidence interval with the sandwich standard errors covers the true mean 97.75% of the time, so it also overestimates the standard error.

(g) Explain, theoretically, what is going wrong in part (f) of this problem. This involves some fairly careful thinking about misspecified mean models and what is required for sandwich estimation to be valid.

Solution: Sandwich estimation protects against misspecification of the variance by using empirical estimates. The *bread* part of the sandwich estimator in the case of ordinary least squares is $X^\intercal \hat{\Sigma} X$, where $\hat{\Sigma}_{ii}$ is diagonal, and $\hat{\Sigma}_{ii} = \left(Y_i - X\hat{\beta}\right)^2$. Thus, the diagonal contains empirical residual estimates. Those estimates depend on our mean model, which in this case is incorrect, biased, and inconsistent since the f is nonlinear. Thus, the results of the sandwich estimator will also be biased and inconsistent.

Appendix

Code for fitting models and generating plots is attached.

Simulations for Problem 1

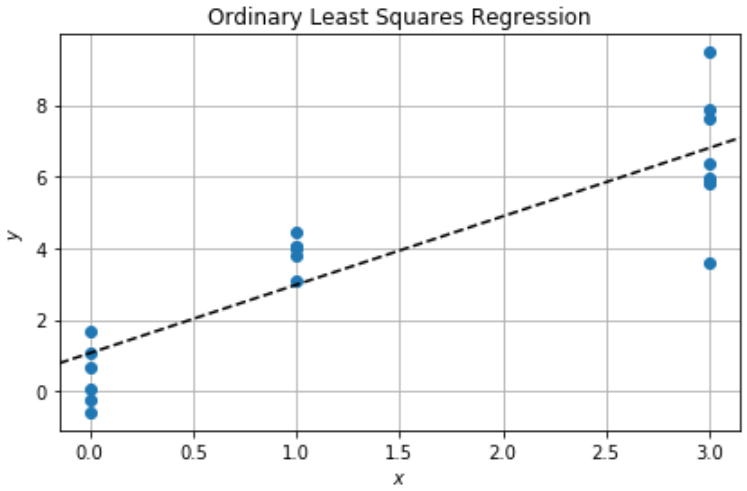
In [1]:
```
import collections
import itertools

import matplotlib.pyplot as plt
import multiprocessing
import numpy as np
import pandas as pd
from scipy import linalg
from scipy import stats

X_VALUES = [0, 1, 3]
MU = np.array([0.6, 3.6, np.nan, 6.8])


def simulate_data(n, random_seed=None):
    np.random.seed(random_seed)
    dist = stats.rv_discrete(values=list(zip(
        *[(x, 1/len(X_VALUES)) for x in X_VALUES])))
    x = dist.rvs(size=(n,))
    while len(np.unique(x)) == 1:
        x = dist.rvs(size=(n,))
    mu = MU[x]
    return x, stats.norm.rvs(loc=mu, scale=1)

def solve_beta_hat(x, y):
    X = np.column_stack((np.ones_like(x), x))
    return linalg.cho_solve(linalg.cho_factor(X.T.dot(X)), np.eye(X.shape[1])).dot
(X.T).dot(y)
```

In [2]:
```
x, y = simulate_data(20, random_seed=2019)
```

In [3]:
```
beta_hat = solve_beta_hat(x, y)
beta_hat
```

Out[3]:
```
array([1.07327318, 1.9097713 ])
```

In [4]:
```
fig = plt.figure(figsize=(6,4))
ax = fig.gca()
ax.grid(True)
ax.plot(x, y, 'o')
xlim_left, xlim_right = ax.get_xlim()
ax.plot([xlim_left, xlim_right],
        np.array([xlim_left, xlim_right])*beta_hat[1] + beta_hat[0], '--k')
ax.set_xlim(xlim_left, xlim_right);
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_title('Ordinary Least Squares Regression')
fig.tight_layout()
fig.savefig('p1_ols.pdf', bbox_inches='tight')
```



In [5]:
```
def _estimate_beta1(n):
    x, y = simulate_data(n)
    return solve_beta_hat(x, y)[1]

def estimate_beta1(num_estimates, n):
    pool = multiprocessing.Pool(4)
    estimates = pool.map(_estimate_beta1, [n]*num_estimates)
    pool.close()
    return np.array(estimates)

def estimate_bias(num_trials):
    n_list = list(range(2, 31, 2)) + list(range(30, 101, 5))
    mean = []
    standard_errors = []
    for n in n_list:
        estimates = estimate_beta1(num_trials, n)
        mean.append(np.mean(estimates))
        standard_errors.append(np.sqrt(np.var(estimates, ddof=1)))
    return pd.DataFrame(collections.OrderedDict([
        ('$n$', n_list),
        ('$\\hat{\\beta}_1$', mean),
        ('$\\hat{\\sigma}$', standard_errors),
    ]))

#bias_simulations = estimate_bias(1000000)
#bias_simulations.to_pickle('p1_simulations.pkl')
bias_simulations = pd.read_pickle('p1_simulations.pkl')
bias_simulations
```

Out[5]:

	n	$\hat{\beta}_1$	$\hat{\sigma}$
0	2	2.222531	1.115256
1	4	2.119261	0.752701
2	6	2.058301	0.536735
3	8	2.028134	0.408784
4	10	2.014462	0.332531
5	12	2.008274	0.284690
6	14	2.005376	0.253067
7	16	2.003600	0.231102
8	18	2.002070	0.213511
9	20	2.001587	0.200356
10	22	2.001091	0.189500
11	24	2.001169	0.180668
12	26	2.000937	0.172259
13	28	2.000748	0.165419
14	30	2.000182	0.159203
15	30	2.000505	0.159288
16	35	2.000618	0.146462
17	40	2.000300	0.136427
18	45	2.000245	0.127994
19	50	2.000259	0.121297
20	55	2.000184	0.115340
21	60	2.000254	0.110308
22	65	2.000014	0.105664
23	70	2.000246	0.101802
24	75	2.000068	0.098277
25	80	2.000033	0.094885
26	85	2.000091	0.092020
27	90	2.000172	0.089272
28	95	1.999983	0.087027
29	100	1.999924	0.084749

In [6]:
```
def partition(n, k = None, m = None):
    k = n if k is None else k
    m = n if m is None else m
    if m is None or m >= n:
        yield [n]

    for f in range(n-1 if m >= n else m, 0, -1):
        if f*(k - 1) < n - f:
            break
        for p in partition(n-f, k - 1, f):
            yield [f] + p


def multiset_permuation(xs):
    current_permutation = len(xs)*[None]
    counts = collections.Counter(xs)

    def yield_permuation(j):
        if j == len(xs):
            yield tuple(current_permutation)
            return

        for x in counts:
            if counts[x] <= 0:
                continue
            counts[x] -= 1
            current_permutation[j] = x
            for permuation in yield_permuation(j + 1):
                yield permuation
            counts[x] += 1

    return yield_permuation(0)

def enumerate_multinomial_support(n, k):
    for p in partition(n, k):
        unordered_draw = [0]*(k - len(p)) + p
        for x in multiset_permuation(unordered_draw):
            yield x
```

In [7]:
```
def solve_expected_beta(n):
    support = enumerate_multinomial_support(n, 3)
    weighted_sum = np.zeros(2)
    total_prob = 0
    for sample in support:
        if np.isin(n, sample):
            continue
        x = np.hstack([count*[X_VALUES[i]] for i, count in enumerate(sample)])
        y = MU[x.astype(np.int)]
        expected_beta = solve_beta_hat(x, y)
        prob = stats.multinomial.pmf(sample, n, p = np.ones(len(X_VALUES))/len(X_V
ALUES))
        weighted_sum += expected_beta*prob
        total_prob += prob
    return weighted_sum/total_prob

expected_beta = np.array([solve_expected_beta(n) for n in bias_simulations['$n$'
]])
expected_beta
```

Out[7]:
```
array([[1.06666667, 2.22222222],
       [1.04168541, 2.11935822],
       [1.0336912 , 2.05765274],
       [1.02878485, 2.02807713],
       [1.02477543, 2.01456074],
       [1.02144138, 2.00820044],
       [1.01872191, 2.00502319],
       [1.01652181, 2.00331956],
       [1.01473662, 2.00233772],
       [1.01327484, 2.00173226],
       [1.01206379, 2.0013359 ],
       [1.01104808, 2.00106296],
       [1.01018608, 2.00086695],
       [1.00944647, 2.00072128],
       [1.00880554, 2.00060992],
       [1.00880554, 2.00060992],
       [1.00752558, 2.0004236 ],
       [1.00656859, 2.0003117 ],
       [1.00582669, 2.0002391 ],
       [1.00523495, 2.00018928],
       [1.0047521 , 2.00015359],
       [1.00435066, 2.00012714],
       [1.00401168, 2.00010698],
       [1.00372166, 2.00009128],
       [1.0034707 , 2.00007879],
       [1.00325143, 2.00006871],
       [1.0030582 , 2.00006045],
       [1.00288664, 2.00005359],
       [1.00273329, 2.00004784],
       [1.00259541, 2.00004297]])
```

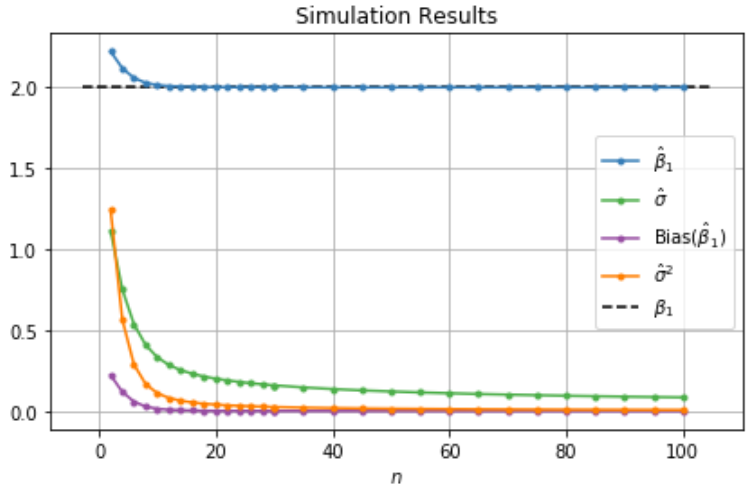
In [8]:
```
bias_simulations['$\\mathbb{E}[\\hat{\\beta}_1]$'] = expected_beta[:,1]
bias_simulations
```

Out[8]:

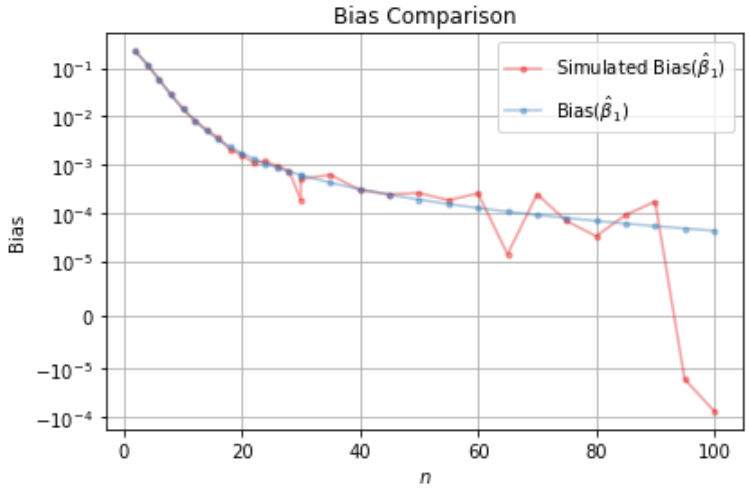
	n	$\hat{\beta}_1$	$\hat{\sigma}$	$\mathbb{E}[\hat{\beta}_1]$
0	2	2.222531	1.115256	2.222222
1	4	2.119261	0.752701	2.119358
2	6	2.058301	0.536735	2.057653
3	8	2.028134	0.408784	2.028077
4	10	2.014462	0.332531	2.014561
5	12	2.008274	0.284690	2.008200
6	14	2.005376	0.253067	2.005023
7	16	2.003600	0.231102	2.003320
8	18	2.002070	0.213511	2.002338
9	20	2.001587	0.200356	2.001732
10	22	2.001091	0.189500	2.001336
11	24	2.001169	0.180668	2.001063
12	26	2.000937	0.172259	2.000867
13	28	2.000748	0.165419	2.000721
14	30	2.000182	0.159203	2.000610
15	30	2.000505	0.159288	2.000610
16	35	2.000618	0.146462	2.000424
17	40	2.000300	0.136427	2.000312
18	45	2.000245	0.127994	2.000239
19	50	2.000259	0.121297	2.000189
20	55	2.000184	0.115340	2.000154
21	60	2.000254	0.110308	2.000127
22	65	2.000014	0.105664	2.000107
23	70	2.000246	0.101802	2.000091
24	75	2.000068	0.098277	2.000079
25	80	2.000033	0.094885	2.000069
26	85	2.000091	0.092020	2.000060
27	90	2.000172	0.089272	2.000054
28	95	1.999983	0.087027	2.000048
29	100	1.999924	0.084749	2.000043

In [9]:
```
with open('p1_simulation_results.tex', 'w') as f:
    f.write(bias_simulations.to_latex(escape=False, index=False))
```

In [10]:
```
fig = plt.figure(figsize=(6, 4))
ax = fig.gca()
ax.grid(True)

ax.plot(bias_simulations['$n$'], bias_simulations['$\\hat{\\beta}_1$'],
        '.-', label='$\\hat{\\beta}_1$', c=plt.cm.Set1(1))
ax.plot(bias_simulations['$n$'], bias_simulations['$\\hat{\\sigma}$'],
        '.-', label='$\\hat{\\sigma}$', c=plt.cm.Set1(2))
ax.plot(bias_simulations['$n$'], bias_simulations['$\\hat{\\beta}_1$'] - 2,
        '.-', label='Bias($\\hat{\\beta}_1$)', c=plt.cm.Set1(3))
ax.plot(bias_simulations['$n$'], bias_simulations['$\\hat{\\sigma}$']**2,
        '.-', label='$\\hat{\\sigma}^2$', c=plt.cm.Set1(4))
ax.hlines(y=2, xmin=ax.get_xlim()[0], xmax=ax.get_xlim()[1],
          linestyles='--', label='$\\beta_1$')
ax.legend()
ax.set_xlabel('$n$')
ax.set_title('Simulation Results')
fig.tight_layout()
fig.savefig('p1_simulation_results.pdf', bbox_inches='tight')
```



In [11]:
```
fig = plt.figure(figsize=(6, 4))
ax = fig.gca()
ax.grid(True)

ax.plot(bias_simulations['$n$'], bias_simulations['$\\hat{\\beta}_1$'] - 2,
        '.-', label='Simulated Bias($\\hat{\\beta}_1$)', c=plt.cm.Set1(0), alpha=
0.4)
ax.plot(bias_simulations['$n$'], bias_simulations['$\mathbb{E}[\\hat{\\beta}_1]$']
 - 2,
        '.-', label='Bias($\\hat{\\beta}_1$)', c=plt.cm.Set1(1), alpha=0.4)
ax.legend()
ax.set_title('Bias Comparison')
ax.set_xlabel('$n$')
ax.set_ylabel('Bias')
ax.set_yscale('symlog', linthreshy=1e-5)
fig.tight_layout()
fig.savefig('p1_bias_comparison.pdf', bbox_inches='tight')
```



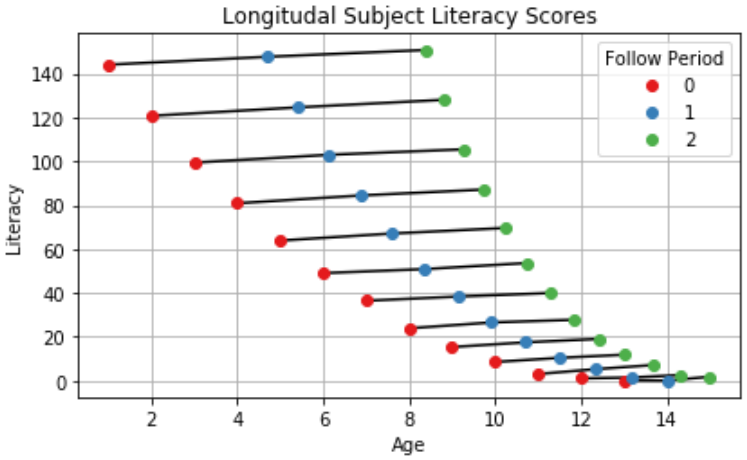
Regression Models and Simulation for Problem 3

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy import linalg
from scipy import stats
```

In [2]:
```
computer_data = pd.read_csv('./computer-data-hw1.csv')

fig = plt.figure(figsize=(6, 3.75))
ax = fig.gca()
ax.grid(True)

for i, subject in enumerate(computer_data['subj'].unique()):
    sub_data = computer_data[computer_data['subj'] == subject]
    ax.plot(sub_data['total.age'], sub_data['literacy'].values, '-k')

for i, follow_per in enumerate(computer_data['follow.per'].unique()):
    sub_data = computer_data[computer_data['follow.per'] == follow_per]
    ax.plot(sub_data['total.age'], sub_data['literacy'],
            'o', color=plt.cm.Set1(i), label='{}'.format(follow_per))

ax.set_xlabel('Age')
ax.set_ylabel('Literacy')
ax.set_title('Longitudal Subject Literacy Scores')
ax.legend(title='Follow Period')
fig.tight_layout()
fig.savefig('p3_data.pdf', bbox_inches='tight')

computer_data.head(n=10)
computer_data[computer_data['follow.per'] == 1].describe()
```

Out[2]:

	subj	base.age	delta.age	total.age	literacy	follow.per
count	13.00000	13.00000	13.000000	13.000000	13.000000	13.0
mean	7.00000	7.00000	2.218935	9.218935	52.103420	1.0
std	3.89444	3.89444	0.879002	3.019735	49.541677	0.0
min	1.00000	1.00000	1.000000	4.698225	0.120457	1.0
25%	4.00000	4.00000	1.514793	6.863905	10.463912	1.0
50%	7.00000	7.00000	2.136095	9.136095	38.377584	1.0
75%	10.00000	10.00000	2.863905	11.514793	84.414795	1.0
max	13.00000	13.00000	3.698225	14.000000	147.606099	1.0



In [3]:
```
def fit_ols(X, y):
    gram_matrix = X.T.dot(X)
    gram_matrix_inv = linalg.cho_solve(linalg.cho_factor(gram_matrix), np.eye(len(
gram_matrix)))
    beta_hat = gram_matrix_inv.dot(X.T).dot(y)
    sigma_2_hat = np.sum(np.square(y - X.dot(beta_hat)))/(len(y) - len(beta_hat))
    return beta_hat, gram_matrix_inv*sigma_2_hat, sigma_2_hat
```

In [4]:
```
def fit_original(data):
    X = np.column_stack((np.ones(len(data)), data[['base.age', 'delta.age']].value
s))
    y = data['literacy'].values
    return fit_ols(X, y)

fit_original(computer_data)
```

Out[4]:
```
(array([133.02690855, -11.9680251 ,   1.24721578]),
 array([[35.38965932, -3.08971504, -4.28560487],
        [-3.08971504,  0.36085295,  0.25406082],
        [-4.28560487,  0.25406082,  1.12990206]]),
 165.83485669258215)
```

In [5]:
```
def fit_alternative(data):
    sub_data = data[data['follow.per'] == 0][['subj', 'literacy']]
    sub_data = pd.merge(data, sub_data.rename(columns={'literacy': 'base.literacy'
}))
    sub_data = sub_data[sub_data['follow.per'] != 0]
    X = sub_data[['delta.age']].values
    y = sub_data['literacy'].values - sub_data['base.literacy'].values
    return fit_ols(X, y)

fit_alternative(computer_data)
```

Out[5]:
```
(array([1.02185154]), array([[0.00105099]]), 0.3850802073163594)
```

Simulations

In [6]:
```
def make_covariates(n):
    X = []
    for i in range(n):
        base_age = stats.uniform.rvs(0, 15)
        X.append([i, base_age, 0, 0])

        for j in range(2):
            delta_age = stats.norm.rvs(3 + j*3, scale=0.5)
            X.append([i, base_age, delta_age, j + 1])

    return pd.DataFrame(X, columns=['subj', 'base.age', 'delta.age', 'follow.per'
])

def make_independent_covariates(n):
    subj = {}
    X = []
    for i in range(n):
        base_age = stats.uniform.rvs(0, 15)
        X.append([i, base_age, 0, 0])
        subj[i] = {'follow_per': 0, 'base_age': base_age}

    for j in range(n*2):
        i = stats.randint.rvs(low=0, high=n)
        subj[i]['follow_per'] += 1
        X.append([i, subj[i]['base_age'], stats.uniform.rvs(0, 10), subj[i]['follo
w_per']])

    return pd.DataFrame(X, columns=['subj', 'base.age', 'delta.age', 'follow.per'
])

def make_response(data, f):
    data = data.copy()
    data['literacy'] = stats.norm.rvs(f(data['base.age']) + 2*data['delta.age'], s
cale=1)
    return data
```

In [7]:
```
np.random.seed(2020)
simulated_data = make_covariates(64)
```

Both Assumptions Violated

In [8]:
```
beta_hat_original_estimates = []
beta_hat_alternative_estimates = []

for i in range(100):
    response_data = make_response(simulated_data, lambda x: 200*np.exp(-x))
    beta_hat, beta_hat_variance, sigma_2_hat = fit_original(response_data)
    beta_hat_original_estimates.append(beta_hat)

    beta_hat, beta_hat_variance, sigma_2_hat = fit_alternative(response_data)
    beta_hat_alternative_estimates.append(beta_hat)


np.mean(beta_hat_original_estimates, 0), np.mean(beta_hat_alternative_estimates, 0
)
```

Out[8]:
```
(array([31.96619861, -3.20229314,  2.16353098]), array([2.00158068]))
```

Linear f but Dependent Covariates

Only the Part (b) condition is now violated.

In [9]:
```
beta_hat_original_estimates = []
beta_hat_alternative_estimates = []

for i in range(100):
    response_data = make_response(simulated_data, lambda x: 200 - 12*x)
    beta_hat, beta_hat_variance, sigma_2_hat = fit_original(response_data)
    beta_hat_original_estimates.append(beta_hat)

    beta_hat, beta_hat_variance, sigma_2_hat = fit_alternative(response_data)
    beta_hat_alternative_estimates.append(beta_hat)


np.mean(beta_hat_original_estimates, 0), np.mean(beta_hat_alternative_estimates, 0
)
```

Out[9]:
```
(array([200.00853446, -11.9993114 ,   1.99779186]), array([1.99509547]))
```

Non-linear f but Independent Covariates

Only the Part (a) condition is violated now.

In [10]:
```
beta_hat_original_estimates = []
beta_hat_alternative_estimates = []

np.random.seed(2019)
for i in range(1024):
    simulated_data = make_independent_covariates(64)
    response_data = make_response(simulated_data, lambda x: 200*np.exp(-x))
    beta_hat, beta_hat_variance, sigma_2_hat = fit_original(response_data)
    beta_hat_original_estimates.append(beta_hat)

    beta_hat, beta_hat_variance, sigma_2_hat = fit_alternative(response_data)
    beta_hat_alternative_estimates.append(beta_hat)

np.mean(beta_hat_original_estimates, 0), np.mean(beta_hat_alternative_estimates, 0
)
```

Out[10]:
```
(array([47.92338712, -4.63105486,  1.99852915]), array([2.00052851]))
```

Standard Error Simulations

In [11]:
```
beta_hat_estimates = []
beta_hat_l_variances = []
is_covered = []
is_covered_sandwich = []

np.random.seed(2019)
for i in range(2048):
    simulated_data = make_independent_covariates(64)
    response_data = make_response(simulated_data, lambda x: 200*np.exp(-x))
    beta_hat, beta_hat_variance, sigma_2_hat = fit_original(response_data)

    X = np.column_stack((np.ones(len(response_data)), response_data[['base.age',
'delta.age']].values))
    y = response_data['literacy']

    gram_inverse = beta_hat_variance/sigma_2_hat
    sandwich_variance = X.T.dot(np.diag(np.square(y - X.dot(beta_hat)))).dot(X)
    sandwich_variance = gram_inverse.dot(sandwich_variance).dot(gram_inverse)

    beta_hat_estimates.append(beta_hat)
    beta_hat_l_variances.append(beta_hat_variance[2,2])
    is_covered.append(np.abs(beta_hat[2] - 2) <= stats.norm.ppf(0.975)*np.sqrt(bet
a_hat_variance[2,2]))
    is_covered_sandwich.append(np.abs(beta_hat[2] - 2) <= stats.norm.ppf(0.975)*np
.sqrt(sandwich_variance[2,2]))

np.sum(is_covered)/len(is_covered), np.sum(is_covered_sandwich)/len(is_covered_san
dwich)
```

Out[11]:
```
(0.98095703125, 0.9775390625)
```