STAT/BIOST 571: Homework 3

Philip Pham

February 12, 2019

Problem 1: Quasilikelihood and semiparametric methods for the general linear model (14 points)

This question examines the effect of different correlation structures, designs, and sample sizes in fitting a general linear model in a quasi-likelihood and semiparametric framework. It is also an exercise in writing code systematically; please take care to break the required programming into small tasks, and write individual functions to do each of these tasks. Please write all code "by hand", using matrix algebra and simple moment-based estimators. You may find the `mvtnorm` *package helpful.*

For the marginal model

$$E(Y_{ij}|x_{ij}) = \beta_0 + \beta_1 x_{ij},$$

consider estimation by weighted least squares, where the cluster weights are the inverse of the estimated cluster covariance matrix. Calculate quasi-likelihood standard errors as if your assumed form of the covariance matrix is known to be correct (even if, in actuality, you have assumed the wrong form of the covariance) and semi-parametric standard errors using the sandwich estimator that accounts for clustering. All of the notation follows the lecture notes.

Throughout, the following are true in the data-generating mechanism

- $\beta_0 = 0$, $\beta_1 = 0.5$
- $\mathbf{Y}_i|\boldsymbol{X}_i \sim N(\boldsymbol{X}_i\boldsymbol{\beta}, \sigma^2\boldsymbol{R}_i)$ *with* $\sigma^2 = 1$.

The factors that will vary are

- *The number of clusters is 15, 30, or 60*
- *The design:*
  - *Design I has* $m_i = 4$*, for all clusters. In each cluster, we see* $\{x_{i1}, x_{i2}, x_{i3}, x_{i4}\} = \{7, 10, 13, 16\}$
  - *Design II has* $m_i = 3$ *for all clusters. We see equal numbers of clusters with* $\{x_{i1}, x_{i2}, x_{i3}\} = \{7, 10, 13\}, \{7, 10, 16\}, \{7, 13, 16\}$, *or* $\{10, 13, 16\}$
- *The true covariance and the assumed covariance matrices are of the form* $\sigma^2\boldsymbol{R}_i$*:*

n	Design	True Corr	$\mathbf{SD}(\hat{\beta}_1)$ Assumed Corr Uncor	Exch	Expon	$\mathbf{E}(\widehat{\mathbf{SE}}_{\mathbf{1,QL}})$ Assumed Corr Uncor	Exch	Expon	$\mathbf{E}(\widehat{\mathbf{SE}}_{\mathbf{1,sand}})$ Assumed Corr Uncor	Exch	Expon
15	I	Exchangeable $\rho = 0.5$	0.03849	0.02849	0.03787	0.03761	0.02756	0.0369	0.02572	0.02572	0.02616
15	I	Exchangeable $\rho = 0.9$	0.03849	0.01466	0.02461	0.03698	0.01376	0.02319	0.0115	0.0115	0.01205
15	I	Exponential $\rho = 0.5$	0.03849	0.03174	0.03799	0.03766	0.03088	0.0371	0.03619	0.03619	0.03577
15	I	Exponential $\rho = 0.9$	0.03849	0.01735	0.0246	0.03703	0.01633	0.02321	0.02021	0.02021	0.01998
15	II	Exchangeable $\rho = 0.5$	0.04472	0.0345	0.04029	0.04408	0.03371	0.03952	0.03402	0.03092	0.0313
15	II	Exchangeable $\rho = 0.9$	0.04472	0.01904	0.02504	0.04352	0.01819	0.02398	0.02594	0.01408	0.01454
15	II	Exponential $\rho = 0.5$	0.04472	0.03654	0.04025	0.04412	0.03581	0.03954	0.03911	0.03748	0.03737
15	II	Exponential $\rho = 0.9$	0.04472	0.02066	0.02492	0.04355	0.01976	0.02386	0.02825	0.01931	0.01907
30	I	Exchangeable $\rho = 0.5$	0.02722	0.01974	0.02682	0.02679	0.01933	0.02635	0.01869	0.01869	0.01908
30	I	Exchangeable $\rho = 0.9$	0.02722	0.00954	0.0163	0.02654	0.00918	0.01572	0.00837	0.00837	0.0088
30	I	Exponential $\rho = 0.5$	0.02722	0.02219	0.02686	0.0268	0.02178	0.02642	0.02614	0.02614	0.02589
30	I	Exponential $\rho = 0.9$	0.02722	0.01157	0.01624	0.02655	0.01116	0.01567	0.01459	0.01459	0.01444
30	II	Exchangeable $\rho = 0.5$	0.03152	0.02378	0.02819	0.03122	0.02345	0.02785	0.02475	0.02245	0.02273
30	II	Exchangeable $\rho = 0.9$	0.03152	0.01201	0.01598	0.03104	0.01172	0.0156	0.01902	0.01018	0.01051
30	II	Exponential $\rho = 0.5$	0.03152	0.02538	0.02818	0.03122	0.02506	0.02786	0.0283	0.02717	0.02705
30	II	Exponential $\rho = 0.9$	0.03152	0.01329	0.01592	0.03106	0.01297	0.01554	0.02067	0.01395	0.01377
60	I	Exchangeable $\rho = 0.5$	0.01925	0.01377	0.01894	0.01909	0.01363	0.01877	0.0134	0.0134	0.01372
60	I	Exchangeable $\rho = 0.9$	0.01925	0.00641	0.01103	0.01901	0.00629	0.01082	0.00599	0.00599	0.00629
60	I	Exponential $\rho = 0.5$	0.01925	0.01557	0.01895	0.0191	0.01543	0.0188	0.01884	0.01884	0.01865
60	I	Exponential $\rho = 0.9$	0.01925	0.00792	0.01101	0.01902	0.00778	0.01082	0.01052	0.01052	0.0104
60	II	Exchangeable $\rho = 0.5$	0.02225	0.01661	0.01983	0.02212	0.01648	0.01969	0.01779	0.01615	0.01635
60	II	Exchangeable $\rho = 0.9$	0.02225	0.00798	0.01065	0.02206	0.00787	0.01051	0.01368	0.00731	0.00755
60	II	Exponential $\rho = 0.5$	0.02225	0.0178	0.01983	0.02213	0.01768	0.0197	0.02032	0.01953	0.01944
60	II	Exponential $\rho = 0.9$	0.02225	0.00895	0.01065	0.02206	0.00883	0.0105	0.01486	0.01001	0.00989

Table 1: Table of standard deviations of the estimated slopes and average of model-based and sandwich-based standard error estimates

- *For the true covariance, consider exchangeable and exponential correlation structures, with $\rho = 0.5$ or $\rho = 0.9$ (distances between observations in the exponential model based on x_{ij}).*
- *For the assumed covariance, consider these and additionally the uncorrelated homoscedastic covariance. Any covariance parameters should be estimated using moment-based methods.*

Solution: See the results in Table 1. Correct standard errors are found when the assumed correlation structure agrees with the true correlation structure when using GLS (columns 2 and 3). Ignoring the correlation within the clusters overestimates the standard error (first column). When incorrectly assuming exponential correlation, the standard error is overestimated. When incorrectly assuming exchangeable correlation, the standard error is underestimated.

Quasi-likelihood doesn't help very much when the correlation structure is misspecified. The standard error estimates are almost identical to just using GLS. The standard errors are slightly closer to those of the sandwich estimator, so there is some insignificant gain.

Comparing the sandwich estimation with GLS when the assumed correlation is correct, one sees that the standard error estimates are underestimated for smaller n. For $n = 60$, the estimates are very good. When the correlation structure is misspecified, sandwich estimation comes closest to the actual the standard error, especially when n is large (verified numerically, results not shown). It follows that misspecifying the correlation structure as exchangeable or exponential doesn't affect the standard error of $\hat{\beta}_1$ very much.

This is also the case when assuming no correlation with design I. However, when using design II, where not all the clusters have identical covariates, assuming no correlation increases the standard error of our estimator significantly. This effect is even more pronounced when there is more correlation ($\rho = 0.9$ versus $\rho = 0.5$).

Details of the calculation are described below, and code is in the appendix.

Let X_i be the covariates for each cluster with a column of 1s prepended. Let y_i be the cluster responses.

We first estimate β with the ordinary least squares (OLS) estimator, $\hat{\beta}_{\text{OLS}} = \left(X^\intercal X\right)^{-1} X^\intercal y$, where X is the concatenation of the cluster covariates and y is the concatenation of the cluster responses.

We can get the covariances for $\hat{\beta}$ with

$$\operatorname{cov}\left(\hat{\beta}\right) = \left(\sum_{i=1}^{n} X_i^\intercal W X_i\right)^{-1} \left(\sum_{i=1}^{n} X_i^\intercal W \hat{\Sigma} W X_i\right) \left(\sum_{i=1}^{n} X_i^\intercal W X_i\right)^{-1}.$$

For the different estimation methods and assumed covariances, we vary the form of $\hat{\Sigma}$ and W, both of which are assumed to be the same for all clusters.

To obtain W, we first estimate σ^2 with

$$\hat{\sigma}^2 = \frac{1}{\sum_{i=1}^{n} m_i - 2} \sum_{i=1}^{n} \left(y_i - X_i \hat{\beta}_{\text{OLS}}\right)^\intercal \left(y_i - X_i \hat{\beta}_{\text{OLS}}\right). \tag{1}$$

Let the standardized OLS residuals for each cluster be $\tilde{\epsilon}_i = \frac{y_i - X_i \hat{\beta}_{\text{OLS}}}{\hat{\sigma}}$.

If we assume that the correlation structure is exchangeable, we estimate

$$\hat{\rho}_{\text{exch}} = \frac{1}{\sum_{i=1}^{n} \sum_{j=1}^{m_i - 1} j} \sum_{i=1}^{n} \sum_{j=1}^{m_i - 1} \sum_{k=j+1}^{m_i} \left(\tilde{\epsilon}_i \tilde{\epsilon}_i^\intercal\right)_{jk}, \tag{2}$$

so $W_{ii}^{-1} = 1$ for all i and $W_{ij}^{-1} = \hat{\rho}_{\text{exch}}$ for all $i \neq j$.

If we assume that the correlation structure is exponential, we estimate

$$\hat{\rho}_{\text{expon}} = \frac{1}{\sum_{i=1}^{n} \sum_{j=1}^{m_i - 1} j} \sum_{i=1}^{n} \sum_{j=1}^{m_i - 1} \left(\tilde{\epsilon}_i \tilde{\epsilon}_i^\intercal\right)_{j,j+1}, \tag{3}$$

so $W_{ij}^{-1} = \hat{\rho}_{\text{expon}}^{|j-i|}$ for any i and j.

If we don't assume any correlation structure, $W = I$.

Now that we know W, we can estimate β with

$$\hat{\beta} = \frac{1}{\sum_{i=1}^{n} m_i} \sum_{i=1}^{n} m_i \left(X_i^\intercal W X_i\right)^{-1} X_i^\intercal W y_i. \tag{4}$$

By default, for the general linear model, we would have that $\hat{\Sigma} = W^{-1}$, in which case, we have that

$$\operatorname{cov}\left(\hat{\beta}\right) = \left(\sum_{i=1}^{n} X_i^\intercal W X_i\right)^{-1}. \tag{5}$$

In the quasi-likelihood model, we have an additional dispersion factor α, so $\hat{\Sigma} = \hat{\alpha} W^{-1}$, where

$$\hat{\alpha} = \frac{1}{\sum_{i=1}^{n} m_i - 2} \left(y - X\hat{\beta} \right)^{\mathsf{T}} \left(y - X\hat{\beta} \right), \tag{6}$$

which results in the covariance

$$\operatorname{cov}\left(\hat{\beta}\right) = \hat{\alpha} \left(\sum_{i=1}^{n} X_i^{\mathsf{T}} W X_i \right)^{-1}. \tag{7}$$

For the sandwich estimator, we use the empirical estimate

$$\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - X_i\hat{\beta} \right)^{\mathsf{T}} \left(y_i - X_i\hat{\beta} \right). \tag{8}$$

Problem 2: Efficiency of OLS for linear models with correlated data (6 points)

Review the example on slides 2.34 – 2.35, which can also be found on pages 60 – 62 of the Diggle et al. textbook. We will generalize this example by considering the mean model

$$E(Y_{ij}) = \beta_0 + \beta_1 x_j$$

for arbitrary $\boldsymbol{x} = (x_1, x_2, \ldots, x_5)$ *that is the same for all subjects, but which may or may not be equal to* $\boldsymbol{t} = (-2, -1, 0, 1, 2)$ *(as is the case in the original version of the example). Note that the correlation structure is still determined based on* t*, as in the original example, but now the mean model contains* x *rather than* t*.*

(a) *Derive a general expression for the relative efficiency of OLS compared to the optimal GLS in estimating* β_0 *and* β_1 *in this problem. Your formula should be valid for a homoscedastic exponential covariance matrix with arbitrary* ρ *and for arbitrary* $\boldsymbol{x}$*. That is, derive a general version of the expressions on the bottom of 2.44 that is valid for any choice of covariates. Note that it is acceptable for your solution to be written using matrix notation and matrix algebra.*

Solution: Let there be n subjects. Each subject i has m_i observations $x_i = \begin{pmatrix} x_{i1} & x_{i2} & \ldots & x_{im_i} \end{pmatrix}^{\mathsf{T}}$. Let X_i be the $m_i \times 2$ covariate matrix, where $X_{ij1} = 1$ and $X_{ij2} = x_{ij}$. Let X be the $\sum_{i=1}^{n} m_i \times 2$ matrix obtained by stacking $X_1, X_2, \ldots, X_n$.

The true model is $Y_{ij} = \beta_0 + \beta_1 x_{ij} + \epsilon_{ij}$. Let the obvervations between subjects be independent with each other. Let Σ_i denote the covariance structure for within-subject observations, that is $\Sigma_{ijk} = \operatorname{cov}(\epsilon_{ij}, \epsilon_{ik})$. Let Σ be the $\sum_{i=1}^{n} m_i \times \sum_{i=1}^{n} m_i$ block diagonal matrix

$$\Sigma = \begin{pmatrix} \Sigma_1 & & & \\ & \Sigma_2 & & \\ & & \ddots & \\ & & & \Sigma_n \end{pmatrix}. \tag{9}$$

Let Y_i be the vector of observations for subject i. Let Y be obtained by concatenating the $Y_1, Y_2, \ldots, Y_n$. If $\beta = \begin{pmatrix}\beta_0 & \beta_1\end{pmatrix}^\intercal$, we can write $Y = X\beta + \epsilon$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \Sigma)$.
The OLS estimate for is

$$\hat{\beta}_{\text{OLS}} = \left(X^\intercal X\right)^{-1} X^\intercal Y = \beta + \left(X^\intercal X\right)^{-1} X^\intercal \epsilon, \tag{10}$$

which has covariance

$$\begin{aligned}\operatorname{cov}\left(\hat{\beta}_{\text{OLS}}\right) &= \left(X^\intercal X\right)^{-1} X^\intercal \operatorname{cov}(\epsilon) X \left(X^\intercal X\right)^{-1} \\ &= \left(X^\intercal X\right)^{-1} X^\intercal \Sigma X \left(X^\intercal X\right)^{-1}.\end{aligned} \tag{11}$$

The optimal GLS estimate, which can be derived by maximizing likelihood is

$$\hat{\beta}_{\text{GLS}} = \left(X^\intercal \Sigma^{-1} X\right)^{-1} X^\intercal \Sigma^{-1} Y = \beta + \left(X^\intercal \Sigma^{-1} X\right)^{-1} X^\intercal \Sigma^{-1} \epsilon, \tag{12}$$

which has covariance

$$\begin{aligned}\operatorname{cov}\left(\hat{\beta}_{\text{GLS}}\right) &= \left(X^\intercal \Sigma^{-1} X\right)^{-1} X^\intercal \Sigma^{-1} \operatorname{cov}(\epsilon) \Sigma^{-1} X \left(X^\intercal \Sigma^{-1} X\right)^{-1} \\ &= \left(X^\intercal \Sigma^{-1} X\right)^{-1}.\end{aligned} \tag{13}$$

Much of these matrix multiplications can be written as summations:

$$\begin{aligned}X^\intercal X &= \sum_{i=1}^n X_i^\intercal X_i \\ X^\intercal \Sigma X &= \sum_{i=1}^n X_i^\intercal \Sigma_i X_i \\ X^\intercal \Sigma^{-1} X &= \sum_{i=1}^n X_i^\intercal \Sigma_i^{-1} X_i.,\end{aligned}$$

which is probably computationally faster if n is very large.
In any case, using Equations 11 and 13, we can calculate relative efficiency

$$\begin{aligned}e\left(\hat{\beta}_0\right) &= \left[\left(X^\intercal \Sigma^{-1} X\right)^{-1}\right]_{11} \Big/ \left[\left(X^\intercal X\right)^{-1} X^\intercal \Sigma X \left(X^\intercal X\right)^{-1}\right]_{11} \\ e\left(\hat{\beta}_1\right) &= \left[\left(X^\intercal \Sigma^{-1} X\right)^{-1}\right]_{22} \Big/ \left[\left(X^\intercal X\right)^{-1} X^\intercal \Sigma X \left(X^\intercal X\right)^{-1}\right]_{22}\end{aligned}$$

by taking the corresponding entries of the covariance matrix.

(b) *Reproduce the lines in the table on 2.35 that pertain to β_1 for the following choices of covariate vectors*

$$\begin{aligned}\boldsymbol{x} &= (-2, -1, 0, 1, 2) \\ \boldsymbol{x} &= (-1, -2, 0, 2, 1) \\ \boldsymbol{x} &= (0, -1, 1, 3, 2) \\ \boldsymbol{x} &= (0, -1, 1, 5, 2)\end{aligned}$$

x	ρ Value	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99
(-2, -1, 0, 1, 2)	$e(\hat{\beta}_0)$	0.9978	0.9917	0.983	0.9729	0.9631	0.9554	0.9521	0.9558	0.9701	0.9961
	$e(\hat{\beta}_1)$	0.9969	0.9893	0.9797	0.97	0.9615	0.9554	0.9522	0.9522	0.9554	0.9608
(-1, -2, 0, 2, 1)	$e(\hat{\beta}_0)$	0.9978	0.9917	0.983	0.9729	0.9631	0.9554	0.9521	0.9558	0.9701	0.9961
	$e(\hat{\beta}_1)$	0.9959	0.9818	0.9554	0.9154	0.8621	0.7974	0.7249	0.6486	0.5726	0.507
(0, -1, 1, 3, 2)	$e(\hat{\beta}_0)$	0.9972	0.9888	0.9758	0.9596	0.9432	0.9302	0.9247	0.931	0.9541	0.9942
	$e(\hat{\beta}_1)$	0.9959	0.9818	0.9554	0.9154	0.8621	0.7974	0.7249	0.6486	0.5726	0.507
(0, -1, 1, 5, 2)	$e(\hat{\beta}_0)$	0.9949	0.9817	0.9636	0.9445	0.9281	0.9178	0.9169	0.9279	0.9541	0.9943
	$e(\hat{\beta}_1)$	0.9911	0.9644	0.9206	0.8626	0.794	0.7194	0.6432	0.5689	0.4992	0.4416

Table 2: Efficiency results comparing OLS with GLS.

Solution: Results can be seen Table 2. Code is in the appendix.

(c) *Explain the key differences between the relative efficiencies you just calculated. Phrase your answers in a manner that will be understandable by a quantitavely sophisticated non-statistician (e.g., an epidemiologist collaborator).*

Solution: If there's not much correlation, i.e. small ρ, there's not much efficiency gain in using GLS over OLS.

For β_1, the efficiency gain does depend on the observed covariates, however. Recall that the estimated for β_1 can be written as a sum of weighted pairwise slopes. The weight is higher for for pairs where x_{ij_1} and x_{ij_2} are further part because the effect size becomes bigger relative to the noise. Similarly, when the errors are positively correlated, more of the variance is explained away, so we are more confident about the slope estimate β_1.

These two factors interact multiplicatively. Thus, when

$$\mathbf{x} \in \{(-1, -2, 0, 2, 1), (0, -1, 1, 3, 2), (0, -1, 1, 5, 2)\},$$

there are pairs of observations where the difference in covariates is large and the correlation is high. GLS is able to leverage this to produce a more efficient estimator. In particular when $\mathbf{x} = (0, -1, 1, 5, 2)$, there are neighboring observations where the difference in covariates can be quite large, leading to the relatively most efficient estimator when using GLS.

Appendix

Code is attached on subsequent pages.

Quasilikelihood and semiparametric methods for the general linear model

In [1]:

```
import enum
import functools
import itertools
import multiprocessing
from typing import Callable, List, NamedTuple, Sequence, Tuple

import numpy as np
import pandas as pd
from scipy import stats
from scipy import linalg

BETA = np.array([0., 0.5], dtype=np.float64)
DESIGN_CLUSTERS = {
    'I': [[7, 10, 13, 16]],
    'II': [[7, 10, 13], [7, 10, 16], [7, 13, 16], [10, 13, 16]],
}
NUM_CLUSTERS = [15, 30, 60]
WITHIN_CLUSTER_CORRELATIONS = [0.5, 0.9]

CorrelationStructure = enum.Enum(
    'CorrelationStructure',
    'NONE EXCHANGEABLE EXPONENTIAL')

EstimationMethod = enum.Enum(
    'EstimationMethod',
    'GLS QL Sandwich')
```

In [2]:
```
class Experiment(NamedTuple('Experiment', [
    ('beta', np.array),
    ('error_variance', float),
    ('num_clusters', Sequence[Tuple[np.array, np.array]]),
    ('clusters', Sequence[np.array]),
    ('within_cluster_correlation', float),
    ('within_cluster_correlation_structure', CorrelationStructure),
])):
    """Encapsulates parameters for the data generating mechanism."""

    def sample_clusters(self) -> List[Tuple[np.array, np.array]]:
        return [self._sample_cluster() for _ in range(self.num_clusters)]

    def _sample_cluster(self) -> Tuple[np.array, np.array]:
        covariates = self._sample_cluster_covariates()
        covariates = np.column_stack((np.ones(len(covariates)), covariates))
        covariance = self._make_within_cluster_covariance(len(covariates))
        response = stats.multivariate_normal(
            mean=np.matmul(covariates, self.beta), cov=covariance).rvs()
        return covariates, response

    def _sample_cluster_covariates(self) -> np.array:
        return self.clusters[np.random.choice(len(self.clusters))]

    def _make_within_cluster_covariance(self, cluster_size):
        correlation = np.eye(cluster_size)
        if self.within_cluster_correlation_structure == CorrelationStructure.EXCHA
NGEABLE:
            correlation[correlation == 0] = self.within_cluster_correlation
        elif self.within_cluster_correlation_structure == CorrelationStructure.EXP
ONENTIAL:
            for i in range(cluster_size):
                for j in range(i + 1, cluster_size):
                    correlation[i, j] = correlation[j, i] = np.power(
                        self.within_cluster_correlation, np.abs(j - i))
        return self.error_variance*correlation

    @classmethod
    def from_template(
        cls,
        clusters,
        num_clusters,
        within_cluster_correlation,
        within_cluster_correlation_structure) -> 'Experiment':
        assert len(set([len(cluster) for cluster in clusters])) == 1,\
            'Clusters must be the same size.'

        return cls(beta=BETA,
                   clusters=clusters,
                   error_variance=1.,
                   num_clusters=num_clusters,
                   within_cluster_correlation=within_cluster_correlation,
                   within_cluster_correlation_structure=within_cluster_correlation
_structure)
```

In [3]:

```
def sum_dict(acc, result):
    if type(acc) == dict:
        return {key: sum_dict(value, result[key]) for key, value in acc.items()}
    return acc + result

def divide_dict(results, d):
    if type(results) == dict:
        return {key: divide_dict(value, d) for key, value in results.items()}
    return results/d
```

In [4]:
```
def estimate_rho(epsilon_hat):
    covariance = np.outer(epsilon_hat, epsilon_hat)
    rho_exchangeable = 0.
    rho_exponential = 0.
    for i in range(len(covariance)):
        for j in range(i + 1, len(covariance[i])):
            rho_exchangeable += covariance[i, j]
            if j - i == 1:
                rho_exponential += covariance[i, j]
    rho_exchangeable /= (np.square(covariance.shape[0]) - covariance.shape[0])/2
    rho_exponential /= covariance.shape[0] - 1
    return rho_exchangeable, rho_exponential

def make_correlation_matrices(clusters, beta_hat, sigma_2_hat):
    rho_exchangeable = 0.
    rho_exponential = 0.
    for X, y in clusters:
        cluster_rho_exchangeable, cluster_rho_exponential = estimate_rho(
            (y - X.dot(beta_hat))/np.sqrt(sigma_2_hat))
        rho_exchangeable += cluster_rho_exchangeable
        rho_exponential += cluster_rho_exponential

    rho_exchangeable /= len(clusters)
    rho_exponential /= len(clusters)

    correlation_matrices = []
    for X, y in clusters:
        exchangeable_matrix = np.eye(len(y))
        exchangeable_matrix[exchangeable_matrix == 0] = rho_exchangeable
        exponential_matrix = np.eye(len(y))
        for i in range(len(y) - 1):
            for j in range(i + 1, len(y)):
                exponential_matrix[i, j] = exponential_matrix[j, i] = np.power(rho
_exponential, j - i)
        correlation_matrices.append({
            CorrelationStructure.NONE.name: np.eye(len(y)),
            CorrelationStructure.EXCHANGEABLE.name: exchangeable_matrix,
            CorrelationStructure.EXPONENTIAL.name: exponential_matrix,
        })

    return correlation_matrices

def estimate_beta_hats(clusters, correlation_matrices):
    def estimate_beta_hat(X, y, correlation_matrix):
        weight = linalg.cho_solve(
            linalg.cho_factor(correlation_matrix), np.eye(len(correlation_matrix
)))
        gram_matrix = linalg.cho_factor(X.T.dot(weight).dot(X))
        return linalg.cho_solve(gram_matrix, X.T.dot(weight).dot(y))

    beta_hats = [
        {
            key: estimate_beta_hat(X, y, inv_weight)
            for key, inv_weight in inv_weights.items()
        } for (X, y), inv_weights in zip(clusters, correlation_matrices)
    ]
    return divide_dict(functools.reduce(sum_dict, beta_hats), len(beta_hats))

def estimate_covariance(clusters,
                        correlation_matrices,
                        method,
                        beta_hat):
    if method != EstimationMethod.Sandwich:
```

```
        covariance = np.zeros((len(beta_hat), len(beta_hat)))
        dispersion_factor = 0.
        total = 0.
        for (X, y), correlation_matrix in zip(clusters, correlation_matrices):
            weight = linalg.cho_solve(
                linalg.cho_factor(correlation_matrix), np.eye(len(correlation_matr
ix)))
            covariance += X.T.dot(weight).dot(X)
            dispersion_factor += np.sum(np.square(y - X.dot(beta_hat)))
            total += len(y)
        covariance = linalg.cho_solve(linalg.cho_factor(covariance), np.eye(len(be
ta_hat)))
        dispersion_factor /= total - len(beta_hat)
        return covariance if method == EstimationMethod.GLS else covariance*disper
sion_factor
    # Sandwich estimation.
    bread = np.zeros((len(beta_hat), len(beta_hat)))
    meat = np.zeros((len(beta_hat), len(beta_hat)))
    for (X, y), correlation_matrix in zip(clusters, correlation_matrices):
        weight = linalg.cho_solve(
                linalg.cho_factor(correlation_matrix), np.eye(len(correlation_matr
ix)))
        bread += X.T.dot(weight).dot(X)
        epsilon_hat = y - X.dot(beta_hat)
        meat += X.T.dot(weight).dot(np.outer(epsilon_hat, epsilon_hat)).dot(weight
).dot(X)
    bread = linalg.cho_solve(linalg.cho_factor(bread), np.eye(len(beta_hat)))
    return bread.dot(meat).dot(bread)

def run_experiment(experiment, estimate_beta=False):
    clusters = experiment.sample_clusters()
    X = np.vstack([X for X, _ in clusters])
    y = np.hstack([y for _, y in clusters])

    gram_matrix_ols = X.T.dot(X)

    beta_hat_ols = linalg.cho_solve(linalg.cho_factor(gram_matrix_ols), X.T.dot(y
))
    sigma_2_hat_ols = np.sum(np.square(y - X.dot(beta_hat_ols)))/(len(y) - len(bet
a_hat_ols))

    correlation_matrices = make_correlation_matrices(
        clusters,
        beta_hat_ols,
        sigma_2_hat_ols)
    beta_hats = estimate_beta_hats(clusters, correlation_matrices)

    if estimate_beta:
        return beta_hats

    return {
        method.name: {
            correlation_structure: np.sqrt(estimate_covariance(
                clusters,
                [matrix_dict[correlation_structure] for matrix_dict in correlation
_matrices],
                method,
                beta_hat)[1, 1])
            for correlation_structure, beta_hat in beta_hats.items()
        }
        for method in EstimationMethod
    }

def run_experiments(experiment, num_trials):
```

```python
    pool = multiprocessing.Pool(4)
    results = pool.map(run_experiment, [experiment]*num_trials)
    results = functools.reduce(sum_dict, results)
    return divide_dict(results, num_trials)
```

In [5]:
```python
experiments = [
    Experiment.from_template(design, num_clusters, correlation_coefficient, correl
ation_structure)
    for design, num_clusters, correlation_coefficient, correlation_structure
    in
    itertools.product(
       [DESIGN_CLUSTERS['I'], DESIGN_CLUSTERS['II']],
       NUM_CLUSTERS,
       WITHIN_CLUSTER_CORRELATIONS,
       [CorrelationStructure.EXCHANGEABLE, CorrelationStructure.EXPONENTIAL])
]
```

In [6]:
```python
def index_experiment(experiment):
    return (experiment.num_clusters,
            [k for k, v in DESIGN_CLUSTERS.items() if experiment.clusters == v][0
],
            experiment.within_cluster_correlation_structure.name,
            experiment.within_cluster_correlation)

simulation_results = pd.DataFrame(
    index=pd.MultiIndex.from_product(
        [NUM_CLUSTERS, DESIGN_CLUSTERS.keys(),
         [CorrelationStructure.EXCHANGEABLE.name, CorrelationStructure.EXPONENTIAL
.name],
         WITHIN_CLUSTER_CORRELATIONS,
        ],
        names=['$n$', 'Design', 'Correlation structure', 'Correlation']),
    columns=pd.MultiIndex.from_product(
        [[value.name for value in EstimationMethod],
         [value.name for value in CorrelationStructure]],
        names=['Estimator', 'Assumed correlation']
    ))
```

In [7]:
```
for experiment in experiments:
    simulation_results.loc[index_experiment(experiment)] = (
        pd.DataFrame.from_dict(run_experiments(experiment, 2048), orient='index').
stack())
simulation_results
```

Out[7]:

			Estimator	GLS			QL	
			Assumed correlation	NONE	EXCHANGEABLE	EXPONENTIAL	NONE	EXCHANGE/
n	Design	Correlation structure	Correlation					
15	II	EXCHANGEABLE	0.5	0.0446405	0.0345911	0.0401501	0.0440066	0.033
			0.9	0.0446405	0.0190978	0.0248836	0.0433367	0.018
		EXPONENTIAL	0.5	0.0446405	0.0366441	0.0402589	0.0441175	0.035
			0.9	0.0446405	0.020748	0.0250146	0.0433865	0.019
	I	EXCHANGEABLE	0.5	0.03849	0.0285347	0.0379085	0.0376814	0.02
			0.9	0.03849	0.0146857	0.0246354	0.0370149	0.013
		EXPONENTIAL	0.5	0.03849	0.0317697	0.0380393	0.0377623	0.031
			0.9	0.03849	0.0173972	0.0246799	0.0370737	0.016
30	II	EXCHANGEABLE	0.5	0.0314797	0.0237564	0.0281589	0.0313385	0.023
			0.9	0.0314797	0.0120085	0.0159871	0.0311532	0.011
		EXPONENTIAL	0.5	0.0314797	0.0253448	0.0281349	0.0313774	0.025
			0.9	0.0314797	0.0133119	0.0160227	0.031173	0.013
	I	EXCHANGEABLE	0.5	0.0272166	0.0196814	0.0268216	0.0269009	0.019
			0.9	0.0272166	0.00948955	0.0162356	0.0266928	0.0091
		EXPONENTIAL	0.5	0.0272166	0.0221441	0.0268359	0.026916	0.021
			0.9	0.0272166	0.011534	0.0161774	0.0267059	0.01
60	II	EXCHANGEABLE	0.5	0.0222499	0.0166306	0.0198637	0.0222153	0.016
			0.9	0.0222499	0.00799161	0.010729	0.0221372	0.0078
		EXPONENTIAL	0.5	0.0222499	0.0178156	0.0198458	0.0222337	0.017
			0.9	0.0222499	0.00897048	0.0107243	0.0221481	0.0088
	I	EXCHANGEABLE	0.5	0.019245	0.0137411	0.0189355	0.0191361	0.013
			0.9	0.019245	0.00639107	0.0110212	0.0190774	0.0062
		EXPONENTIAL	0.5	0.019245	0.0155442	0.0189305	0.0191339	0.01
			0.9	0.019245	0.00789485	0.010963	0.0190793	0.0077

In [8]:
```
import os

if not os.path.isdir('simulation_results'):
    os.mkdir('simulation_results')

for key, values in simulation_results.iterrows():
    file_name = '-'.join(map(str, key)).replace('.', '_')
    with open('simulation_results/{}.tex'.format(file_name), 'w') as f:
        f.write(' & '.join(map(lambda v: str(np.round(v, decimals=5)), values.valu
es)))
```

Efficiency of OLS for linear models with correlated data

In [1]:
```
import collections
import numpy as np
import pandas as pd
from scipy import linalg
```

In [2]:
```
def make_covariates(n, cluster_covariates):
    covariates = np.tile(cluster_covariates, n)
    return np.column_stack((np.ones_like(covariates), covariates))
```

In [3]:
```
def make_exponential_correlation_matrix(m, rho):
    correlation_matrix = np.eye(m)
    for i in range(m - 1):
        for j in range(i + 1, m):
            correlation_matrix[i, j] = correlation_matrix [j, i] = np.power(rho, j
- i)
    return correlation_matrix
```

In [4]:
```
def compute_efficiency(n, cluster_covariates, rho):
    X = make_covariates(n, cluster_covariates)
    sigma = linalg.block_diag(*(
        [make_exponential_correlation_matrix(len(cluster_covariates), rho)]*n))

    gram_matrix_inv = linalg.cho_solve(linalg.cho_factor(X.T.dot(X)), np.eye(X.sha
pe[1]))
    covariance_ols = gram_matrix_inv.dot(X.T.dot(sigma).dot(X)).dot(gram_matrix_in
v)

    weights = linalg.cho_solve(linalg.cho_factor(sigma), np.eye(X.shape[0]))
    covariance_gls = linalg.cho_solve(
        linalg.cho_factor(X.T.dot(weights).dot(X)), np.eye(X.shape[1]))

    return np.diag(covariance_gls)/np.diag(covariance_ols)
```

In [5]:
```
N = 10
CLUSTER_COVARIATES = [
    [-2,-1,0,1,2],
    [-1,-2,0,2,1],
    [0,-1,1,3,2],
    [0,-1,1,5,2],
]
RHO = np.hstack((np.linspace(0.1, 0.9, 9), [0.99]))

efficiency_results = collections.OrderedDict([
    (
        str(cluster_covariates),
        {
            str(np.round(rho, 2)): compute_efficiency(N, cluster_covariates, rho)
            for rho in RHO
        },
    )
    for cluster_covariates in CLUSTER_COVARIATES
])
```

In [6]:

```python
efficiency_table = pd.DataFrame(
    index=pd.MultiIndex.from_product([
        map(str, CLUSTER_COVARIATES),
        ['$e(\\hat{\\beta}_0)$', '$e(\\hat{\\beta}_1)$'],
    ], names=['$x$', 'Value']),
    columns=pd.Series(map(lambda rho: str(np.round(rho, 2)), RHO), name='$\\rho$'
),
)
for cluster_covariates, values in efficiency_results.items():
    for rho, efficiencies in values.items():
        efficiency_table[rho][cluster_covariates, '$e(\\hat{\\beta}_0)$'] = effici
encies[0]
        efficiency_table[rho][cluster_covariates, '$e(\\hat{\\beta}_1)$'] = effici
encies[1]

with open('p2_efficiencies.tex', 'w') as f:
    f.write(efficiency_table.to_latex(
        escape=False,
        float_format=lambda f: str(np.round(f, 4))).replace('[', '(').replace(']',
')'))

efficiency_table
```

Out[6]:

ρ		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.9
x	**Value**										
[-2, -1, 0, 1, 2]	$e(\hat{\beta}_0)$	0.99776	0.991707	0.982965	0.972888	0.963082	0.955424	0.952117	0.955846	0.970123	0.99611
	$e(\hat{\beta}_1)$	0.996874	0.9893	0.979685	0.969955	0.961538	0.955424	0.952221	0.952221	0.955424	0.96081
[-1, -2, 0, 2, 1]	$e(\hat{\beta}_0)$	0.99776	0.991707	0.982965	0.972888	0.963082	0.955424	0.952117	0.955846	0.970123	0.99611
	$e(\hat{\beta}_1)$	0.995921	0.98184	0.955424	0.915402	0.862069	0.797438	0.724923	0.648636	0.572575	0.50702
[0, -1, 1, 3, 2]	$e(\hat{\beta}_0)$	0.997183	0.988849	0.975751	0.959626	0.943244	0.930221	0.924682	0.93103	0.954102	0.99415
	$e(\hat{\beta}_1)$	0.995921	0.98184	0.955424	0.915402	0.862069	0.797438	0.724923	0.648636	0.572575	0.50702
[0, -1, 1, 5, 2]	$e(\hat{\beta}_0)$	0.994938	0.981695	0.96362	0.944474	0.928059	0.917849	0.91686	0.927875	0.954054	0.99432
	$e(\hat{\beta}_1)$	0.991123	0.964352	0.920638	0.862553	0.793975	0.719406	0.643193	0.568935	0.499199	0.44157