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):
    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]:

```
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]:

```
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	EXCHANGEA
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)))
```