Iterative methods for solving GEE

In [1]:
```
import numpy as np
import pandas as pd
from scipy import linalg
from typing import List, NamedTuple

class CovarianceParameters(NamedTuple('CovarianceParameters', [
    ('alpha', np.float64),
    ('sigma2', np.float64),
])):
    def make_correlation_matrix(self, size):
        correlation_matrix = np.eye(size)
        for i in range(size - 1):
            for j in range(i + 1, size):
                correlation_matrix[i, j] = correlation_matrix[j, i] = np.power(sel
f.alpha, j - i)
        return correlation_matrix

    def make_covariance_matrix(self, size):
        return self.make_correlation_matrix(size)*self.sigma2

class Cluster(NamedTuple('Cluster', [
    ('X', np.array),
    ('y', np.array),
    ('covariance', np.array),
])):
    """Cluster covariates, response, and covariance structure."""
```

In [2]:

```
def estimate_covariance_parameters(clusters, beta):
    def _estimate_covariance_parameters(X, y, beta):
        epsilon = y - X.dot(beta)
        covariance_matrix = np.outer(epsilon, epsilon)
        sigma2 = np.diag(covariance_matrix)
        rho = [covariance_matrix[i, i + 1] for i in range(len(covariance_matrix) -
1)]
        return sigma2, rho

    sigma2 = []
    rho = []
    for cluster in clusters:
        cluster_sigma2, cluster_rho = _estimate_covariance_parameters(
            cluster.X, cluster.y, beta)
        sigma2.extend(cluster_sigma2)
        rho.extend(cluster_rho)
    sigma2 = np.mean(sigma2)
    return CovarianceParameters(alpha=np.mean(rho)/sigma2, sigma2=sigma2)

def estimate_beta(clusters: List[Cluster]):
    """Estimate beta under the assumption that clusters."""
    cluster_weights = [
        linalg.cho_solve(linalg.cho_factor(cluster.covariance), np.eye(len(cluster
.y))) for cluster in clusters
    ]
    projected_X = np.sum([
        cluster.X.T.dot(weights).dot(cluster.X)
        for cluster, weights in zip(clusters, cluster_weights)
    ], 0)
    projected_y = np.sum([
        cluster.X.T.dot(weights).dot(cluster.y)
        for cluster, weights in zip(clusters, cluster_weights)
    ], 0)
    beta = linalg.cho_solve(linalg.cho_factor(projected_X), projected_y)

    bread = linalg.cho_solve(linalg.cho_factor(projected_X), np.eye(len(projected_
X)))
    residuals = [cluster.y - cluster.X.dot(beta) for cluster in clusters]
    sigma = np.mean([np.outer(residual, residual) for residual in residuals], 0)
    meat = np.sum([
        cluster.X.T.dot(weights).dot(sigma).dot(weights).dot(cluster.X)
        for cluster, residual, weights in zip(clusters, residuals, cluster_weights
)
    ], 0)
    return beta, bread.dot(meat).dot(bread)

def update_clusters(clusters, covariance_parameters):
    return [
        Cluster(X=cluster.X, y=cluster.y,
                covariance=covariance_parameters.make_covariance_matrix(len(cluste
r.y)))
        for cluster in clusters]
```

In [3]:
```
orthodont_data = pd.read_csv('orthodont.csv')
orthodont_data = orthodont_data.set_index('Subject')
orthodont_data.head(8)
```

Out[3]:

Subject	distance	age	Sex
M01	26.0	8	Male
M01	25.0	10	Male
M01	29.0	12	Male
M01	31.0	14	Male
M02	21.5	8	Male
M02	22.5	10	Male
M02	23.0	12	Male
M02	26.5	14	Male

In [4]:
```
def make_covariates(data_frame):
    age = (data_frame['age'] - 8).values
    is_female = (data_frame['Sex'] == 'Female').values.astype(np.float64)
    return np.column_stack((
        np.ones(len(data_frame)),
        age,
        is_female,
        age*is_female,
    ))

def make_response(data_frame):
    return data_frame['distance'].values
```

In [5]:
```
def make_table(beta, beta_covariance):
    std_errors = np.sqrt(np.diag(beta_covariance))
    return pd.DataFrame(list(zip(beta, std_errors)),
                        columns=['Estimate', 'Standard error'],
                        index=['$\\beta_0$', '$\\beta_1$', '$\\beta_2$', '$\\beta_
3$'])
```

Initially, we assume no covariance.

In [6]:
```
clusters = [
    Cluster(
        X=make_covariates(orthodont_data.loc[i]),
        y=make_response(orthodont_data.loc[i]),
        covariance=np.eye(len(orthodont_data.loc[i])),
    )
    for i in orthodont_data.index
]

beta, beta_covariance = estimate_beta(clusters)
with open('beta0_estimate.tex', 'w') as f:
    f.write(make_table(beta, beta_covariance).to_latex(escape=False))
```

In each iteration, we estimate covariance parameters with $\hat{\beta}$. We compute a new correlation structure for each cluster, and use it to get a better estimate for β.

In [7]:
```
def _update_estimates(clusters, beta):
    covariance_parameters = estimate_covariance_parameters(clusters, beta)
    clusters = update_clusters(clusters, covariance_parameters)
    return covariance_parameters, clusters, estimate_beta(clusters)

previous_beta, (covariance_parameters, clusters, (beta, beta_covariance)) = (
    beta, _update_estimates(clusters, beta))

with open('beta1_estimate.tex', 'w') as f:
    f.write(make_table(beta, beta_covariance).to_latex(escape=False))
covariance_parameters
```

Out[7]:
```
CovarianceParameters(alpha=0.6113624114825353, sigma2=4.905158354377104)
```

In [8]:
```
while np.sum(np.abs(previous_beta - beta)) > 1e-12:
    previous_beta, (covariance_parameters, clusters, (beta, beta_covariance)) = (
        beta, _update_estimates(clusters, beta))
    print(beta, covariance_parameters)
    print(np.sqrt(np.diag(beta_covariance)))
```

```
[22.75026233  0.76945687 -1.55885764 -0.28569215] CovarianceParameters(alpha=0.61
3518833989525, sigma2=4.910599805251229)
[0.26779157 0.04369856 0.41954876 0.06846249]
[22.7502655   0.76945666 -1.55886113 -0.28569188] CovarianceParameters(alpha=0.61
35308146270477, sigma2=4.91065185077109)
[0.2677919  0.04369862 0.41954928 0.06846258]
[22.75026552  0.76945666 -1.55886115 -0.28569188] CovarianceParameters(alpha=0.61
35308813311675, sigma2=4.910652141021696)
[0.2677919  0.04369862 0.41954928 0.06846258]
[22.75026552  0.76945666 -1.55886115 -0.28569188] CovarianceParameters(alpha=0.61
35308817025588, sigma2=4.910652142637751)
[0.2677919  0.04369862 0.41954928 0.06846258]
[22.75026552  0.76945666 -1.55886115 -0.28569188] CovarianceParameters(alpha=0.61
35308817046278, sigma2=4.910652142646753)
[0.2677919  0.04369862 0.41954928 0.06846258]
[22.75026552  0.76945666 -1.55886115 -0.28569188] CovarianceParameters(alpha=0.61
35308817046404, sigma2=4.91065214264681)
[0.2677919  0.04369862 0.41954928 0.06846258]
```

The final, converged estimates are below.

In [9]:
```
with open('beta_final_estimate.tex', 'w') as f:
    f.write(make_table(beta, beta_covariance).to_latex(escape=False))
```